

Anytime pmnk-landscapes

Generated by Doxygen 1.9.2



|   |          |
|---|----------|
| <b>1 Hierarchical Index</b>                   | <b>1</b> |
| 1.1 Class Hierarchy                           | 1        |
| <b>2 Class Index</b>                          | <b>3</b> |
| 2.1 Class List                                | 3        |
| <b>3 File Index</b>                           | <b>5</b> |
| 3.1 File List                                 | 5        |
| <b>4 Class Documentation</b>                  | <b>7</b> |
| 4.1 pmnk::EPS Struct Reference                | 7        |
| 4.1.1 Detailed Description                    | 7        |
| 4.1.2 Member Function Documentation           | 7        |
| 4.1.2.1 operator()                            | 7        |
| 4.2 pmnk::GASolution Class Reference          | 8        |
| 4.2.1 Detailed Description                    | 9        |
| 4.2.2 Constructor & Destructor Documentation  | 9        |
| 4.2.2.1 GASolution() [1/2]                    | 9        |
| 4.2.2.2 GASolution() [2/2]                    | 9        |
| 4.2.3 Member Function Documentation           | 9        |
| 4.2.3.1 fitness()                             | 9        |
| 4.2.3.2 set_fitness()                         | 10       |
| 4.2.3.3 set_objv()                            | 10       |
| 4.3 pmnk::GSEMO Class Reference               | 10       |
| 4.3.1 Detailed Description                    | 11       |
| 4.3.2 Constructor & Destructor Documentation  | 11       |
| 4.3.2.1 GSEMO() [1/6]                         | 11       |
| 4.3.2.2 GSEMO() [2/6]                         | 12       |
| 4.3.2.3 GSEMO() [3/6]                         | 12       |
| 4.3.2.4 GSEMO() [4/6]                         | 13       |
| 4.3.2.5 GSEMO() [5/6]                         | 13       |
| 4.3.2.6 GSEMO() [6/6]                         | 14       |
| 4.3.3 Member Function Documentation           | 14       |
| 4.3.3.1 run()                                 | 14       |
| 4.3.3.2 solutions()                           | 14       |
| 4.4 pmnk::hvobj< T > Class Template Reference | 15       |
| 4.4.1 Detailed Description                    | 15       |
| 4.4.2 Member Function Documentation           | 16       |
| 4.4.2.1 remove()                              | 16       |
| 4.5 pmnk::IBEA Class Reference                | 16       |
| 4.5.1 Detailed Description                    | 17       |
| 4.5.2 Constructor & Destructor Documentation  | 17       |
| 4.5.2.1 IBEA() [1/6]                          | 17       |

|  |    |
|--|----|
| 4.5.2.2 IBEA() [2/6]   | 17 |
| 4.5.2.3 IBEA() [3/6]   | 18 |
| 4.5.2.4 IBEA() [4/6]   | 18 |
| 4.5.2.5 IBEA() [5/6]   | 19 |
| 4.5.2.6 IBEA() [6/6]   | 19 |
| 4.5.3 Member Function Documentation                                | 20 |
| 4.5.3.1 run()  | 20 |
| 4.5.3.2 solutions()  | 21 |
| 4.6 pmnk::IHD< R > Struct Template Reference                       | 21 |
| 4.6.1 Detailed Description   | 21 |
| 4.6.2 Constructor & Destructor Documentation                       | 22 |
| 4.6.2.1 IHD()  | 22 |
| 4.6.3 Member Function Documentation                                | 22 |
| 4.6.3.1 operator()   | 22 |
| 4.7 pmnk::KWayTournamentSelection< RNG > Struct Template Reference | 23 |
| 4.7.1 Detailed Description   | 23 |
| 4.7.2 Constructor & Destructor Documentation                       | 23 |
| 4.7.2.1 KWayTournamentSelection()                                  | 23 |
| 4.7.3 Member Function Documentation                                | 24 |
| 4.7.3.1 operator()   | 24 |
| 4.8 pmnk::NPointCrossover< RNG > Struct Template Reference         | 24 |
| 4.8.1 Detailed Description   | 25 |
| 4.8.2 Constructor & Destructor Documentation                       | 25 |
| 4.8.2.1 NPointCrossover()  | 25 |
| 4.8.3 Member Function Documentation                                | 25 |
| 4.8.3.1 operator()   | 25 |
| 4.9 pmnk::PLS Class Reference                                      | 26 |
| 4.9.1 Detailed Description   | 27 |
| 4.9.2 Constructor & Destructor Documentation                       | 27 |
| 4.9.2.1 PLS() [1/6]  | 27 |
| 4.9.2.2 PLS() [2/6]  | 27 |
| 4.9.2.3 PLS() [3/6]  | 28 |
| 4.9.2.4 PLS() [4/6]  | 28 |
| 4.9.2.5 PLS() [5/6]  | 29 |
| 4.9.2.6 PLS() [6/6]  | 29 |
| 4.9.3 Member Function Documentation                                | 30 |
| 4.9.3.1 non_visited_solutions()                                    | 30 |
| 4.9.3.2 run()  | 30 |
| 4.9.3.3 solutions()  | 30 |
| 4.10 pmnk::RMNKEval Class Reference                                | 31 |
| 4.10.1 Detailed Description  | 31 |
| 4.11 pmnk::Solution Class Reference                                | 32 |

|  |           |
|--|-----------|
| 4.11.1 Detailed Description  | 33        |
| 4.11.2 Constructor & Destructor Documentation  | 33        |
| 4.11.2.1 Solution() [1/4]  | 33        |
| 4.11.2.2 Solution() [2/4]  | 33        |
| 4.11.2.3 Solution() [3/4]  | 34        |
| 4.11.2.4 Solution() [4/4]  | 34        |
| 4.11.3 Member Function Documentation   | 34        |
| 4.11.3.1 decision_vector()   | 34        |
| 4.11.3.2 dominance()   | 34        |
| 4.11.3.3 eval()  | 35        |
| 4.11.3.4 neighborhood_solutions()  | 35        |
| 4.11.3.5 objective_vector()  | 35        |
| 4.11.3.6 operator=() [1/2]   | 36        |
| 4.11.3.7 operator=() [2/2]   | 36        |
| 4.11.3.8 operator[]()  | 36        |
| 4.11.3.9 random_solution()   | 37        |
| 4.11.3.10 size()   | 37        |
| 4.11.3.11 uniform_bit_flip_solution()  | 38        |
| 4.11.4 Friends And Related Function Documentation  | 38        |
| 4.11.4.1 operator<<  | 38        |
| 4.12 pmnk::UniformCrossover< RNG > Struct Template Reference                                   | 39        |
| 4.12.1 Detailed Description  | 39        |
| 4.12.2 Constructor & Destructor Documentation  | 39        |
| 4.12.2.1 UniformCrossover()  | 39        |
| 4.12.3 Member Function Documentation   | 40        |
| 4.12.3.1 operator()()  | 40        |
| 4.13 pmnk::UniformMutation< RNG > Struct Template Reference                                    | 40        |
| 4.13.1 Detailed Description  | 41        |
| 4.13.2 Constructor & Destructor Documentation  | 41        |
| 4.13.2.1 UniformMutation()   | 41        |
| 4.13.3 Member Function Documentation   | 41        |
| 4.13.3.1 operator()()  | 41        |
| <b>5 File Documentation</b>  | <b>43</b> |
| 5.1 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/GSEMO/gsemo.hpp File Reference  | 43        |
| 5.1.1 Detailed Description   | 43        |
| 5.2 gsemo.hpp  | 44        |
| 5.3 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp File Reference | 45        |
| 5.3.1 Detailed Description   | 45        |
| 5.4 functor.hpp  | 46        |
| 5.5 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ibea.hpp File Reference    | 47        |
| 5.5.1 Detailed Description   | 48        |

|   |    |
|---|----|
| 5.6 ibea.hpp . . . . .  | 48 |
| 5.7 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/main.cpp File Reference . . . . .            | 51 |
| 5.7.1 Detailed Description . . . . .  | 52 |
| 5.7.2 Macro Definition Documentation . . . . .  | 53 |
| 5.7.2.1 CROSSOVER . . . . .   | 53 |
| 5.7.2.2 INDICATOR . . . . .   | 53 |
| 5.7.2.3 MUTATION . . . . .  | 54 |
| 5.7.2.4 SELECTION . . . . .   | 54 |
| 5.7.3 Enumeration Type Documentation . . . . .  | 54 |
| 5.7.3.1 Crossover . . . . .   | 55 |
| 5.7.3.2 Indicator . . . . .   | 55 |
| 5.7.3.3 Mutation . . . . .  | 55 |
| 5.7.3.4 Selection . . . . .   | 55 |
| 5.7.4 Function Documentation . . . . .  | 55 |
| 5.7.4.1 gsemo() . . . . .   | 55 |
| 5.7.4.2 ibea() . . . . .  | 56 |
| 5.7.4.3 pls() . . . . .   | 57 |
| 5.7.4.4 set_general_options() . . . . .   | 58 |
| 5.7.4.5 set_ibea_crossover_options() . . . . .  | 58 |
| 5.7.4.6 set_ibea_mutation_options() . . . . .   | 58 |
| 5.7.4.7 set_ibea_options() . . . . .  | 59 |
| 5.7.4.8 set_ibea_selection_options() . . . . .  | 59 |
| 5.7.4.9 set_pls_options() . . . . .   | 59 |
| 5.7.4.10 set_positional_arguments() . . . . .   | 60 |
| 5.8 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/PLS/pls.hpp File Reference . . . . .         | 60 |
| 5.8.1 Detailed Description . . . . .  | 61 |
| 5.8.2 Macro Definition Documentation . . . . .  | 61 |
| 5.8.2.1 RUNLOOP . . . . .   | 61 |
| 5.9 pls.hpp . . . . .   | 62 |
| 5.10 rMNKEval.hpp . . . . .   | 64 |
| 5.11 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/solution.hpp File Reference . . . . . | 68 |
| 5.11.1 Detailed Description . . . . .   | 68 |
| 5.12 solution.hpp . . . . .   | 69 |
| 5.13 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/Utils.hpp File Reference . . . . .    | 71 |
| 5.13.1 Detailed Description . . . . .   | 71 |
| 5.13.2 Function Documentation . . . . .   | 71 |
| 5.13.2.1 add_non_dominated() . . . . .  | 71 |
| 5.14 Utils.hpp . . . . .  | 72 |
| 5.15 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/wfg.hpp File Reference . . . . .      | 72 |
| 5.15.1 Detailed Description . . . . .   | 73 |
| 5.15.2 Function Documentation . . . . .   | 74 |
| 5.15.2.1 insert_non_dominated() . . . . .   | 74 |

---

|  |           |
|--|-----------|
| 5.15.2.2 <code>limit_set()</code> . . . . .        | 74        |
| 5.15.2.3 <code>point_hv()</code> . . . . .         | 75        |
| 5.15.2.4 <code>point_hvc()</code> . . . . .        | 75        |
| 5.15.2.5 <code>set_hv()</code> . . . . .           | 76        |
| 5.15.2.6 <code>set_hv_wfg()</code> . . . . .       | 76        |
| 5.15.2.7 <code>weakly_dominates()</code> . . . . . | 77        |
| 5.16 <code>wfg.hpp</code> . . . . .                | 77        |
| <b>Index</b>                                       | <b>83</b> |





# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

|  |    |
|--|----|
| pmnk::EPS . . . . .                            | 7  |
| pmnk::GSEMO . . . . .                          | 10 |
| pmnk::hvobj< T > . . . . .                     | 15 |
| pmnk::IBEA . . . . .                           | 16 |
| pmnk::IHD< R > . . . . .                       | 21 |
| pmnk::KWayTournamentSelection< RNG > . . . . . | 23 |
| pmnk::NPointCrossover< RNG > . . . . .         | 24 |
| pmnk::PLS . . . . .                            | 26 |
| pmnk::RMNKEval . . . . .                       | 31 |
| pmnk::Solution . . . . .                       | 32 |
| pmnk::GASolution . . . . .                     | 8  |
| pmnk::UniformCrossover< RNG > . . . . .        | 39 |
| pmnk::UniformMutation< RNG > . . . . .         | 40 |



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

|   |    |
|---|----|
| <a href="#">pmnk::EPS</a>   |    |
| Additive epsilon indicator . . . . .  | 7  |
| <a href="#">pmnk::GASolution</a>  |    |
| Genetic algorithm solution wrapper (adds a fitness attribute to the <a href="#">Solution</a> class) . . . . .               | 8  |
| <a href="#">pmnk::GSEMO</a>   |    |
| Wrapper class for <a href="#">GSEMO</a> . . . . .   | 10 |
| <a href="#">pmnk::hvobj&lt; T &gt;</a>  |    |
| Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm) . . . . . | 15 |
| <a href="#">pmnk::IBEA</a>  |    |
| Wrapper class for <a href="#">IBEA</a> . . . . .  | 16 |
| <a href="#">pmnk::IHD&lt; R &gt;</a>  |    |
| Hypervolume Based <a href="#">IBEA</a> indicator . . . . .  | 21 |
| <a href="#">pmnk::KWayTournamentSelection&lt; RNG &gt;</a>  |    |
| <a href="#">KWayTournamentSelection</a> operator . . . . .  | 23 |
| <a href="#">pmnk::NPointCrossover&lt; RNG &gt;</a>  |    |
| N-Point crossover operator . . . . .  | 24 |
| <a href="#">pmnk::PLS</a>   |    |
| Wrapper class for <a href="#">PLS</a> . . . . .   | 26 |
| <a href="#">pmnk::RMNKEval</a>  |    |
| Rmnk_landscapes instance evaluator . . . . .  | 31 |
| <a href="#">pmnk::Solution</a>  |    |
| Standard solution class . . . . .   | 32 |
| <a href="#">pmnk::UniformCrossover&lt; RNG &gt;</a>   |    |
| Uniform Crossover operator . . . . .  | 39 |
| <a href="#">pmnk::UniformMutation&lt; RNG &gt;</a>  |    |
| Uniform Mutation operator . . . . .   | 40 |



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

|   |    |
|---|----|
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/ <a href="#">main.cpp</a>  |    |
| Driver program to test the implementation of some search algorithms and simplify the gathering of anytime data relevant to the study of their performance in the context of the pmnk-landscapes problem . . . . . | 51 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/GSEMO/ <a href="#">gsemo.hpp</a>   |    |
| GSEMO (Global Simple Multi-objective Optimizer) algorithm implementation . . . . .  | 43 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ <a href="#">functor.hpp</a>  |    |
| Implementation of IBEA operators (using functors) . . . . .   | 45 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ <a href="#">ibea.hpp</a>   |    |
| IBEA (Indicator Based Evolutionary Algorithm) implementation . . . . .  | 47 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/PLS/ <a href="#">pls.hpp</a>   |    |
| PLS (Pareto Local Search) algorithm implementation . . . . .  | 60 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/ <a href="#">rMNKEval.hpp</a>  |    |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/ <a href="#">solution.hpp</a>  |    |
| Implementation of a solution class . . . . .  | 68 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/ <a href="#">utils.hpp</a>   |    |
| Project Utility functions . . . . .   | 71 |
| /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/ <a href="#">wfg.hpp</a>   |    |
| Implementation of the wfg algorithm to calculate hypervolumes . . . . .   | 72 |



## Chapter 4

# Class Documentation

### 4.1 pmnk::EPS Struct Reference

Additive epsilon indicator.

```
#include <functor.hpp>
```

#### Public Member Functions

- constexpr **EPS** ()=default  
*Construct a new [EPS](#) object.*
- template<typename S = GASolution>  
constexpr double **operator()** (S const &s1, S const &s2) const noexcept  
*Function call operator overload. Implements the indicator functionality.*

#### 4.1.1 Detailed Description

Additive epsilon indicator.

#### 4.1.2 Member Function Documentation

##### 4.1.2.1 operator()()

```
template<typename S = GASolution>  
constexpr double pmnk::EPS::operator() (  
    S const & s1,  
    S const & s2 ) const    [inline], [constexpr], [noexcept]
```

Function call operator overload. Implements the indicator functionality.

### Template Parameters

|          |   |
|----------|---|
| <b>S</b> | The type used to store an genetic algorithm ( <a href="#">IBEA</a> ) solution |
|----------|---|

### Parameters

|           |                             |
|-----------|-----------------------------|
| <i>s1</i> | A solution to be evaluated. |
| <i>s2</i> | A solution to be evaluated. |

### Returns

double The value for the indicator.

The documentation for this struct was generated from the following file:

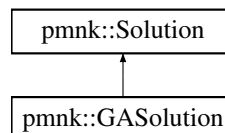
- </home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp>

## 4.2 pmnk::GASolution Class Reference

Genetic algorithm solution wrapper (adds a fitness attribute to the [Solution](#) class)

```
#include <solution.hpp>
```

Inheritance diagram for pmnk::GASolution:



### Public Member Functions

- **GASolution** ()=default  
*Construct a new [GASolution](#) object.*
- **GASolution** ([Solution](#) &&sol, double [fitness](#))  
*Construct a new [GASolution](#) object (move)*
- **GASolution** ([Solution](#) &&sol)  
*Construct a new [GASolution](#) object.*
- constexpr double const & [fitness](#) () const  
*Getter method for the [GASolution](#)'s fitness value.*
- void [set\\_objv](#) (ObjectiveVector &&objv)  
*Set the objv object of the current solution.*
- constexpr void [set\\_fitness](#) (double const [fitness](#))  
*Setter method for the fitness value.*



## Additional Inherited Members

### 4.2.1 Detailed Description

Genetic algorithm solution wrapper (adds a fitness attribute to the [Solution](#) class)

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 GASolution() [1/2]

```
pmnk::GASolution::GASolution (
    Solution && sol,
    double fitness ) [inline]
```

Construct a new [GASolution](#) object (move)

##### Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>sol</i>     | A rvalue reference to a solution.  |
| <i>fitness</i> | The fitness value of the solution. |

#### 4.2.2.2 GASolution() [2/2]

```
pmnk::GASolution::GASolution (
    Solution && sol ) [inline]
```

Construct a new [GASolution](#) object.

##### Parameters

|            |   |
|------------|---|
| <i>sol</i> | A rvalue reference to a solution. (fitness value defaults to 0) |
|------------|---|

### 4.2.3 Member Function Documentation

#### 4.2.3.1 fitness()

```
constexpr double const & pmnk::GASolution::fitness ( ) const [inline], [constexpr]
```

Getter method for the [GASolution](#)'s fitness value.

**Returns**

constexpr double const& The fitness value.

**4.2.3.2 set\_fitness()**

```
constexpr void pmnk::GASolution::set_fitness (
    double const fitness ) [inline], [constexpr]
```

Setter method for the fitness value.

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>fitness</i> | The fitness value. |
|----------------|--------------------|

**4.2.3.3 set\_objv()**

```
void pmnk::GASolution::set_objv (
    ObjectiveVector && objv ) [inline]
```

Set the objv object of the current solution.

**Parameters**

|             |   |
|-------------|---|
| <i>objv</i> | A rvalue reference to the objective vector. |
|-------------|---|

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/[solution.hpp](#)

**4.3 pmnk::GSEMO Class Reference**

Wrapper class for [GSEMO](#).

```
#include <gsemo.hpp>
```

**Public Member Functions**

- `template<typename Str = std::string, typename Ref = ObjectiveVector>`  
[GSEMO](#) (Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)  
*Construct a new [GSEMO](#) object.*
- `template<typename Str = std::string>`  
[GSEMO](#) (Str &&instance, unsigned int const seed, std::ostream &os)

- Construct a new [GSEMO](#) object.
- `template<typename Str = std::string, typename Ref = ObjectiveVector>`  
[GSEMO](#) (Str &&instance, unsigned int const seed, Ref &&ref)  
 Construct a new [GSEMO](#) object.
- `template<typename Str = std::string>`  
[GSEMO](#) (Str &&instance, unsigned int const seed)  
 Construct a new [GSEMO](#) object.
- `template<typename Str = std::string, typename Ref = ObjectiveVector>`  
[GSEMO](#) (Str &&instance, Ref &&ref)  
 Construct a new [GSEMO](#) object.
- `template<typename Str = std::string>`  
[GSEMO](#) (Str &&instance)  
 Construct a new [GSEMO](#) object.
- `std::vector< Solution > const & solutions () const`  
 Getter for the vector of solutions found by this algorithm.
- `void run (std::size_t maxeval)`  
[GSEMO](#) implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

## Public Attributes

- [RMNKEval](#) eval

### 4.3.1 Detailed Description

Wrapper class for [GSEMO](#).

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 GSEMO() [1/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::GSEMO::GSEMO (
    Str && instance,
    unsigned int const seed,
    std::ostream & os,
    Ref && ref ) [inline]
```

Construct a new [GSEMO](#) object.

#### Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">GSEMO</a>                                      |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 4.3.2.2 GSEMO() [2/6]

```
template<typename Str = std::string>
pmnk::GSEMO::GSEMO (
    Str && instance,
    unsigned int const seed,
    std::ostream & os ) [inline]
```

Construct a new [GSEMO](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">GSEMO</a>                                      |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |

## 4.3.2.3 GSEMO() [3/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::GSEMO::GSEMO (
    Str && instance,
    unsigned int const seed,
    Ref && ref ) [inline]
```

Construct a new [GSEMO](#) object.

## Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">GSEMO</a>                                      |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 4.3.2.4 GSEMO() [4/6]

```
template<typename Str = std::string>
pmnk::GSEMO::GSEMO (
    Str && instance,
    unsigned int const seed ) [inline]
```

Construct a new [GSEMO](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">GSEMO</a>                                      |

## 4.3.2.5 GSEMO() [5/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::GSEMO::GSEMO (
    Str && instance,
    Ref && ref ) [inline], [explicit]
```

Construct a new [GSEMO](#) object.

## Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

**Parameters**

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

**4.3.2.6 GSEMO() [6/6]**

```
template<typename Str = std::string>
pmnk::GSEMO::GSEMO (
    Str && instance ) [inline], [explicit]
```

Construct a new [GSEMO](#) object.

**Template Parameters**

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

**Parameters**

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
|-----------------|--|

**4.3.3 Member Function Documentation****4.3.3.1 run()**

```
void pmnk::GSEMO::run (
    std::size_t maxeval ) [inline]
```

[GSEMO](#) implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

**Parameters**

|                |   |
|----------------|---|
| <i>maxeval</i> | The maximum number of evaluations performed by <a href="#">GSEMO</a> (stopping criterion) |
|----------------|---|

**4.3.3.2 solutions()**

```
std::vector< Solution > const & pmnk::GSEMO::solutions ( ) const [inline]
```

Getter for the vector of solutions found by this algorithm.

#### Returns

std::vector<Solution> const& Read-Only reference to a vector of solutions found by the [GSEMO](#) algorithm.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/GSEMO/[gsemo.hpp](#)

## 4.4 pmnk::hvobj< T > Class Template Reference

Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm)

```
#include <wfg.hpp>
```

### Public Types

- using **hv\_type** = T
- using **ovec\_type** = std::vector< hv\_type >
- using **set\_type** = std::vector< ovec\_type >

### Public Member Functions

- constexpr **hvobj** (ovec\_type const &r)
- constexpr **hvobj** ([hvobj](#) const &other)=default
- constexpr **hvobj** ([hvobj](#) &&other) noexcept=default
- constexpr auto **value** () const  
*Get the current hypervolume value.*
- template<typename V >  
constexpr auto **contribution** (V const &v) const  
*Get the contribution of a new vector w.r.t. to the current set.*
- template<typename V >  
constexpr auto **insert** (V &&v)  
*Inserts a new objective vector and returns its contribution.*
- template<typename V >  
constexpr auto [remove](#) (V const &v)

#### 4.4.1 Detailed Description

```
template<typename T>
class pmnk::hvobj< T >
```

Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm)

## 4.4.2 Member Function Documentation

### 4.4.2.1 remove()

```
template<typename T >
template<typename V >
constexpr auto pmnk::hvobj< T >::remove (
    V const & v ) [inline], [constexpr]
```

Removes a objective vector and returns its contribution (i.e. the lost hv) or -1.0 if no objective vector was found.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/wfg.hpp

## 4.5 pmnk::IBEA Class Reference

Wrapper class for [IBEA](#).

```
#include <ibea.hpp>
```

### Public Member Functions

- template<typename Str = std::string, typename Ref = ObjectiveVector>  
[IBEA](#) (Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)  
*Construct a new [IBEA](#) object.*
- template<typename Str = std::string>  
[IBEA](#) (Str &&instance, unsigned int const seed, std::ostream &os)  
*Construct a new [IBEA](#) object.*
- template<typename Str = std::string, typename Ref = ObjectiveVector>  
[IBEA](#) (Str &&instance, unsigned int const seed, Ref &&ref)  
*Construct a new [IBEA](#) object.*
- template<typename Str = std::string>  
[IBEA](#) (Str &&instance, unsigned int const seed)  
*Construct a new [IBEA](#) object.*
- template<typename Str = std::string, typename Ref = ObjectiveVector>  
[IBEA](#) (Str &&instance, Ref &&ref)  
*Construct a new [IBEA](#) object.*
- template<typename Str = std::string>  
[IBEA](#) (Str &&instance)  
*Construct a new [IBEA](#) object.*
- std::vector< [GASolution](#) > const & solutions () const  
*Getter for the vector of solutions found by this algorithm.*
- template<typename I , typename S , typename M , typename C >  
void run (std::size\_t const maxeval, std::size\_t const population\_max\_size, std::size\_t const max\_↵ generations, double const scaling\_factor, I &&indicator, C &&crossover\_method, M &&mutation\_method, S &&selection\_method, bool adaptive)  
*[IBEA](#) implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.*



## Public Attributes

- [RMNKEval](#) eval

### 4.5.1 Detailed Description

Wrapper class for [IBEA](#).

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 IBEA() [1/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::IBEA::IBEA (
    Str && instance,
    unsigned int const seed,
    std::ostream & os,
    Ref && ref ) [inline]
```

Construct a new [IBEA](#) object.

#### Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

#### Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">IBEA</a>                                       |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

#### 4.5.2.2 IBEA() [2/6]

```
template<typename Str = std::string>
pmnk::IBEA::IBEA (
    Str && instance,
    unsigned int const seed,
    std::ostream & os ) [inline]
```

Construct a new [IBEA](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">IBEA</a>                                       |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |

## 4.5.2.3 IBEA() [3/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::IBEA::IBEA (
    Str && instance,
    unsigned int const seed,
    Ref && ref ) [inline]
```

Construct a new [IBEA](#) object.

## Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">IBEA</a>                                       |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 4.5.2.4 IBEA() [4/6]

```
template<typename Str = std::string>
pmnk::IBEA::IBEA (
    Str && instance,
    unsigned int const seed ) [inline]
```

Construct a new [IBEA](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">IBEA</a>                                       |

## 4.5.2.5 IBEA() [5/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::IBEA::IBEA (
    Str && instance,
    Ref && ref ) [inline], [explicit]
```

Construct a new [IBEA](#) object.

## Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">IBEA</a>                                       |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 4.5.2.6 IBEA() [6/6]

```
template<typename Str = std::string>
pmnk::IBEA::IBEA (
    Str && instance ) [inline], [explicit]
```

Construct a new [IBEA](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
|-----------------|--|

### 4.5.3 Member Function Documentation

#### 4.5.3.1 run()

```
template<typename I , typename S , typename M , typename C >
void pmnk::IBEA::run (
    std::size_t const maxeval,
    std::size_t const population_max_size,
    std::size_t const max_generations,
    double const scaling_factor,
    I && indicator,
    C && crossover_method,
    M && mutation_method,
    S && selection_method,
    bool adaptive ) [inline]
```

[IBEA](#) implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

## Template Parameters

|          |  |
|----------|--|
| <i>I</i> | The type used to store an <a href="#">IBEA</a> indicator           |
| <i>S</i> | The type used to store an <a href="#">IBEA</a> selection operator  |
| <i>M</i> | The type used to store and <a href="#">IBEA</a> mutation operator  |
| <i>C</i> | The type used to store and <a href="#">IBEA</a> crossover operator |

## Parameters

|                            |  |
|----------------------------|--|
| <i>maxeval</i>             | The maximum number of evaluations performed by the algorithms (stopping criterion)   |
| <i>population_max_size</i> | The maximum population size  |
| <i>max_generations</i>     | The maximum number of generations  |
| <i>scaling_factor</i>      | The scaling factor   |
| <i>indicator</i>           | The indicator to be used by the <a href="#">IBEA</a> indicator operator  |
| <i>crossover_method</i>    | The crossover method considered by the <a href="#">IBEA</a> mutation operator  |
| <i>mutation_method</i>     | The mutation method considered by the <a href="#">IBEA</a> mutation operator   |
| <i>selection_method</i>    | The selection method considered by the <a href="#">IBEA</a> selection operator   |
| <i>adaptive</i>            | boolean indicative of version of <a href="#">IBEA</a> to be used. If true use adaptive version of (A-IBEA) else use (B-IBEA) |

### 4.5.3.2 solutions()

```
std::vector< GASolution > const & pmnk::IBEA::solutions ( ) const [inline]
```

Getter for the vector of solutions found by this algorithm.

#### Returns

std::vector<Solution> const& Read-Only reference to a vector of solutions found by the [IBEA](#).

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/[ibea.hpp](#)

## 4.6 pmnk::IHD< R > Struct Template Reference

Hypervolume Based [IBEA](#) indicator.

```
#include <functor.hpp>
```

### Public Member Functions

- constexpr [IHD](#) (R &&ref)  
*Construct a new [IHD](#) object.*
- template<typename S = GASolution>  
double [operator\(\)](#) (S const &s1, S const &s2) const  
*Function call operator overload. Implements the indicator functionality.*

### Public Attributes

- R [m\\_ref](#)

### 4.6.1 Detailed Description

```
template<typename R = ObjectiveVector>
struct pmnk::IHD< R >
```

Hypervolume Based [IBEA](#) indicator.

#### Template Parameters

|          |  |
|----------|--|
| <i>R</i> | The <a href="#">IBEA</a> reference point type. |
|----------|--|

## 4.6.2 Constructor & Destructor Documentation

### 4.6.2.1 IHD()

```
template<typename R = ObjectiveVector>
constexpr pmnk::IHD< R >::IHD (
    R && ref ) [inline], [explicit], [constexpr]
```

Construct a new [IHD](#) object.

#### Parameters

|            |   |
|------------|---|
| <i>ref</i> | The reference point used for indicator calculation. |
|------------|---|

## 4.6.3 Member Function Documentation

### 4.6.3.1 operator>()

```
template<typename R = ObjectiveVector>
template<typename S = GASolution>
double pmnk::IHD< R >::operator() (
    S const & s1,
    S const & s2 ) const [inline]
```

Function call operator overload. Implements the indicator functionality.

#### Template Parameters

|          |   |
|----------|---|
| <i>S</i> | The type used to store an genetic algorithm ( <a href="#">IBEA</a> ) solution |
|----------|---|

#### Parameters

|           |                             |
|-----------|-----------------------------|
| <i>s1</i> | A solution to be evaluated. |
| <i>s2</i> | A solution to be evaluated. |

#### Returns

double The value for the indicator.

The documentation for this struct was generated from the following file:

- </home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp>

## 4.7 pmnk::KWayTournamentSelection< RNG > Struct Template Reference

[KWayTournamentSelection](#) operator.

```
#include <functor.hpp>
```

### Public Member Functions

- constexpr [KWayTournamentSelection](#) (std::size\_t const tournament\_size, std::size\_t const matting\_pool\_size, RNG &rng)  
*Construct a new [KWayTournamentSelection](#) object.*
- template<typename S = GASolution>  
std::vector< S > [operator\(\)](#) (std::vector< S > const &population) noexcept  
*Function call operator overload. Implements the selection operator functionality.*

### Public Attributes

- std::size\_t **m\_tournament\_size**
- std::size\_t **m\_matting\_pool\_size**
- RNG **m\_rng**

#### 4.7.1 Detailed Description

```
template<typename RNG>
struct pmnk::KWayTournamentSelection< RNG >
```

[KWayTournamentSelection](#) operator.

##### Template Parameters

|            |  |
|------------|--|
| <i>RNG</i> | The type for the random number generator object. |
|------------|--|

#### 4.7.2 Constructor & Destructor Documentation

##### 4.7.2.1 KWayTournamentSelection()

```
template<typename RNG >
constexpr pmnk::KWayTournamentSelection< RNG >::KWayTournamentSelection (
    std::size_t const tournament_size,
    std::size_t const matting_pool_size,
    RNG & rng ) [inline], [constexpr]
```

Construct a new [KWayTournamentSelection](#) object.

## Parameters

|                          |  |
|--------------------------|--|
| <i>tournament_size</i>   | The size of the tournament (K)                                     |
| <i>matting_pool_size</i> | The the maximum number of individuals allowed in the matting pool. |
| <i>rng</i>               | The random number generator objects.                               |

### 4.7.3 Member Function Documentation

#### 4.7.3.1 operator()

```
template<typename RNG >
template<typename S = GASolution>
std::vector< S > pmnk::KWayTournamentSelection< RNG >::operator() (
    std::vector< S > const & population ) [inline], [noexcept]
```

Function call operator overload. Implements the selection operator functionality.

## Template Parameters

|   |   |
|---|---|
| S | The type used to store an genetic algorithm ( <a href="#">IBEA</a> ) solution |
|---|---|

## Parameters

|                   |   |
|-------------------|---|
| <i>population</i> | The population from which the individuals will be selected. |
|-------------------|---|

## Returns

std::vector<S> The matting pool obtain as a result from the selection of the individuals of the population.

The documentation for this struct was generated from the following file:

- </home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp>

## 4.8 pmnk::NPointCrossover< RNG > Struct Template Reference

N-Point crossover operator.

```
#include <functor.hpp>
```

### Public Member Functions

- constexpr [NPointCrossover](#) (std::size\_t const crossover\_points, double const crossover\_probability, RNG &rng)  
Construct a new [NPointCrossover](#).
- template<typename S = GASolution>  
void [operator\(\)](#) (S &s1, S &s2) noexcept  
Function call operator overload. Implements the crossover operator functionality.



## Public Attributes

- `std::size_t m_crossover_points`
- `double m_crossover_probability`
- `RNG m_rng`
- `std::uniform_real_distribution< double > m_distrib`

### 4.8.1 Detailed Description

```
template<typename RNG>
struct pmnk::NPointCrossover< RNG >
```

N-Point crossover operator.

Template Parameters

|            |  |
|------------|--|
| <i>RNG</i> | The type for the random number generator object. |
|------------|--|

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 NPointCrossover()

```
template<typename RNG >
constexpr pmnk::NPointCrossover< RNG >::NPointCrossover (
    std::size_t const crossover_points,
    double const crossover_probability,
    RNG & rng ) [inline], [constexpr]
```

Construct a new [NPointCrossover](#).

Parameters

|                              |   |
|------------------------------|---|
| <i>crossover_points</i>      | Number of crossover points considered by this operator. |
| <i>crossover_probability</i> | Crossover probability considered by this operator.      |
| <i>rng</i>                   | The random number generator object instance.            |

### 4.8.3 Member Function Documentation

#### 4.8.3.1 operator>()()

```
template<typename RNG >
template<typename S = GASolution>
```

```
void pmnk::NPointCrossover< RNG >::operator() (
    S & s1,
    S & s2 ) [inline], [noexcept]
```

Function call operator overload. Implements the crossover operator functionality.

#### Parameters

|           |                              |
|-----------|------------------------------|
| <i>s1</i> | A solution to be recombined. |
| <i>s2</i> | A solution to be recombined. |

The documentation for this struct was generated from the following file:

- </home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp>

## 4.9 pmnk::PLS Class Reference

Wrapper class for [PLS](#).

```
#include <pls.hpp>
```

### Public Member Functions

- `template<typename Str = std::string, typename Ref = ObjectiveVector>`  
[PLS](#) (Str &&instance, unsigned int seed, std::ostream &os, Ref &&ref)  
*Construct a new [PLS](#) object.*
- `template<typename Str = std::string>`  
[PLS](#) (Str &&instance, unsigned int seed, std::ostream &os)  
*Construct a new [PLS](#) object.*
- `template<typename Str = std::string, typename Ref = ObjectiveVector>`  
[PLS](#) (Str &&instance, unsigned int seed, Ref &&ref)  
*Construct a new [PLS](#) object.*
- `template<typename Str = std::string>`  
[PLS](#) (Str &&instance, unsigned int seed)  
*Construct a new [PLS](#) object.*
- `template<typename Str = std::string, typename Ref = ObjectiveVector>`  
[PLS](#) (Str &&instance, Ref &&ref)  
*Construct a new [PLS](#) object.*
- `template<typename Str = std::string>`  
[PLS](#) (Str &&instance)  
*Construct a new [PLS](#) object.*
- `std::vector< Solution > const solutions () const`  
*Getter for the vector of solutions found by this algorithm.*
- `std::vector< Solution > const non_visited_solutions () const`  
*Getter for the vector of solutions found by this algorithm that were not visited in the process of local search.*
- `void run (std::size_t maxeval, PLSAcceptanceCriterion const acceptance_criterion, PLSExplorationCriterion const neighborhood_exploration)`  
*[PLS](#) implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.*

## Public Attributes

- [RMNKEval](#) eval

### 4.9.1 Detailed Description

Wrapper class for [PLS](#).

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 PLS() [1/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::PLS::PLS (
    Str && instance,
    unsigned int seed,
    std::ostream & os,
    Ref && ref ) [inline]
```

Construct a new [PLS](#) object.

#### Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

#### Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">PLS</a>  |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

#### 4.9.2.2 PLS() [2/6]

```
template<typename Str = std::string>
pmnk::PLS::PLS (
    Str && instance,
    unsigned int seed,
    std::ostream & os ) [inline]
```

Construct a new [PLS](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">PLS</a>  |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |

## 4.9.2.3 PLS() [3/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::PLS::PLS (
    Str && instance,
    unsigned int seed,
    Ref && ref ) [inline]
```

Construct a new [PLS](#) object.

## Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">PLS</a>  |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 4.9.2.4 PLS() [4/6]

```
template<typename Str = std::string>
pmnk::PLS::PLS (
    Str && instance,
    unsigned int seed ) [inline]
```

Construct a new [PLS](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>seed</i>     | The seed used by the pseudo random number generator used in <a href="#">PLS</a>  |

## 4.9.2.5 PLS() [5/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::PLS::PLS (
    Str && instance,
    Ref && ref ) [inline]
```

Construct a new [PLS](#) object.

## Template Parameters

|            |   |
|------------|---|
| <i>Str</i> | the type used to store the instance path                    |
| <i>Ref</i> | the type used to store the reference point of the hvobj obj |

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 4.9.2.6 PLS() [6/6]

```
template<typename Str = std::string>
pmnk::PLS::PLS (
    Str && instance ) [inline]
```

Construct a new [PLS](#) object.

## Template Parameters

|            |  |
|------------|--|
| <i>Str</i> | the type used to store the instance path |
|------------|--|

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
|-----------------|--|

### 4.9.3 Member Function Documentation

#### 4.9.3.1 non\_visited\_solutions()

```
std::vector< Solution > const pmnk::PLS::non_visited_solutions ( ) const [inline]
```

Getter for the vector of solutions found by this algorithm that were not visited in the process of local search.

## Returns

std::vector<Solution> const& Read-Only reference to a vector of non visited solutions produced by [PLS](#).

#### 4.9.3.2 run()

```
void pmnk::PLS::run (
    std::size_t maxeval,
    PLSAcceptanceCriterion const acceptance_criterion,
    PLSExplorationCriterion const neighborhood_exploration ) [inline]
```

[PLS](#) implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

## Parameters

|                                 |   |
|---------------------------------|---|
| <i>maxeval</i>                  | The maximum number of evaluations performed by <a href="#">PLS</a> (stopping criterion) |
| <i>acceptance_criterion</i>     | The <a href="#">PLS</a> algorithm solution acceptance criterion                         |
| <i>neighborhood_exploration</i> | The <a href="#">PLS</a> algorithm solution exploration criterion                        |

#### 4.9.3.3 solutions()

```
std::vector< Solution > const pmnk::PLS::solutions ( ) const [inline]
```

Getter for the vector of solutions found by this algorithm.

**Returns**

std::vector<Solution> const& Read-Only reference to a vector of visited solutions produced by [PLS](#).

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/PLS/[pls.hpp](#)

## 4.10 pmnk::RMNKEval Class Reference

rmnk\_landscapes instance evaluator

```
#include <rmNKEval.hpp>
```

**Public Member Functions**

- **RMNKEval** (const char \*\_fileName)
- void **eval** (std::vector< bool > &\_solution, std::vector< double > &\_objVec)
- unsigned **getM** ()
- unsigned **getN** ()
- unsigned **getK** ()
- double **getRho** ()

**Protected Member Functions**

- virtual void **load** (const char \*\_fileName)
- void **init** ()
- void **loadLinks** (std::fstream &\_file)
- void **loadTables** (std::fstream &\_file)
- double **evalNK** (unsigned \_numObj, std::vector< bool > &\_sol)
- unsigned int **sigma** (unsigned \_numObj, std::vector< bool > &\_sol, int \_i)

**Protected Attributes**

- double **rho**
- unsigned **M**
- unsigned **N**
- unsigned **K**
- double \*\*\* **tables**
- unsigned \*\*\* **links**

### 4.10.1 Detailed Description

rmnk\_landscapes instance evaluator

The documentation for this class was generated from the following file:

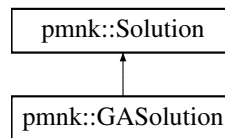
- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/rMNKEval.hpp

## 4.11 pmnk::Solution Class Reference

Standard solution class.

```
#include <solution.hpp>
```

Inheritance diagram for pmnk::Solution:



### Public Member Functions

- [Solution](#) ([Solution](#) const &other)=default  
*Construct a new [Solution](#) object (copy constructor)*
- [Solution](#) ([Solution](#) &&other)=default  
*Construct a new [Solution](#) object (move constructor)*
- [Solution](#) & [operator=](#) ([Solution](#) const &other)=default  
*Defaulted copy assignment operator.*
- [Solution](#) & [operator=](#) ([Solution](#) &&other)=default  
*Defaulted move assignment operator.*
- [Solution](#) ([RMNKEval](#) &rmnk, [DecisionVector](#) const &decision)  
*Construct a new [Solution](#) object.*
- [Solution](#) ([RMNKEval](#) &rmnk, [DecisionVector](#) &&decision)  
*Construct a new [Solution](#) object.*
- [DecisionVector](#) const & [decision\\_vector](#) () const  
*Getter for the solution's decision vector.*
- [ObjectiveVector](#) const & [objective\\_vector](#) () const  
*Getter for the solution's objective vector.*
- [std::size\\_t](#) [size](#) () const noexcept  
*Getter for the solution's decision/objective vector size.*
- [DominanceType](#) [dominance](#) ([Solution](#) const &solution) const  
*Calculate the objective dominance type of this solution with respect to another.*
- [DecisionVector::reference](#) [operator\[\]](#) ([std::size\\_t](#) const i)  
*Array Indexing operator overload. This operator provides a way to access the i-th bit in the solution's decision vector implementation.*
- void [eval](#) ([RMNKEval](#) &rmnk)  
*Wrapper method that calls the RMNK solution evaluator supplied on the solution's decision vector, calculating the respective objective vector.*

### Static Public Member Functions

- [template](#)<typename RNG >  
static [Solution](#) [random\\_solution](#) ([RMNKEval](#) &eval, RNG &generator)  
*Build and evaluate a new random solution object.*
- [template](#)<typename RNG >  
static [Solution](#) [uniform\\_bit\\_flip\\_solution](#) ([RMNKEval](#) &eval, RNG &generator, [Solution](#) const &original)  
*Build and evaluate a new Random solution object that results from a mutation in another solution's decision vector bit representation.*
- static [std::vector](#)< [Solution](#) > [neighborhood\\_solutions](#) ([RMNKEval](#) &eval, [Solution](#) const &original)  
*Calculate all the neighbor solutions of the current one.*



## Protected Attributes

- DecisionVector **m\_decision**
- ObjectiveVector **m\_objective**

## Friends

- std::ostream & [operator<<](#) (std::ostream &os, [Solution](#) const &solution)  
*Extraction operator overload for this object.*

### 4.11.1 Detailed Description

Standard solution class.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 [Solution\(\)](#) [1/4]

```
pmnk::Solution::Solution (
    Solution const & other ) [default]
```

Construct a new [Solution](#) object (copy constructor)

Parameters

|              |   |
|--------------|---|
| <i>other</i> | A const lvalue reference to a solution to be copied |
|--------------|---|

#### 4.11.2.2 [Solution\(\)](#) [2/4]

```
pmnk::Solution::Solution (
    Solution && other ) [default]
```

Construct a new [Solution](#) object (move constructor)

Parameters

|              |  |
|--------------|--|
| <i>other</i> | A rvalue reference to a solution to be moved |
|--------------|--|

#### 4.11.2.3 Solution() [3/4]

```
pmnk::Solution::Solution (
    RMNKEval & rmnk,
    DecisionVector const & decision ) [inline]
```

Construct a new [Solution](#) object.

##### Parameters

|                 |  |
|-----------------|--|
| <i>rmnk</i>     | A lvalue reference to the RMNK instance evaluator. |
| <i>decision</i> | The solution's decision vector.                    |

#### 4.11.2.4 Solution() [4/4]

```
pmnk::Solution::Solution (
    RMNKEval & rmnk,
    DecisionVector && decision ) [inline]
```

Construct a new [Solution](#) object.

##### Parameters

|                 |  |
|-----------------|--|
| <i>rmnk</i>     | A lvalue reference to the RMNK instance evaluator. |
| <i>decision</i> | The solution's decision vector.                    |

### 4.11.3 Member Function Documentation

#### 4.11.3.1 decision\_vector()

```
DecisionVector const & pmnk::Solution::decision_vector ( ) const [inline]
```

Getter for the solution's decision vector.

##### Returns

DecisionVector const& A Read-Only reference to the solution's decision vector.

#### 4.11.3.2 dominance()

```
DominanceType pmnk::Solution::dominance (
    Solution const & solution ) const [inline]
```

Calculate the objective dominance type of this solution with respect to another.

## Parameters

|                 |   |
|-----------------|---|
| <i>solution</i> | A solution whose dominance type of this will be tested against. |
|-----------------|---|

## Returns

DominanceType The solution's dominance type

**4.11.3.3 eval()**

```
void pmnk::Solution::eval (
    RMNKEval & rmnk ) [inline]
```

Wrapper method that calls the RMNK solution evaluator supplied on the solution's decision vector, calculating the respective objective vector.

## Parameters

|             |   |
|-------------|---|
| <i>rmnk</i> | The RMNK instance evaluator instance that provides the method used for solution evaluation. |
|-------------|---|

**4.11.3.4 neighborhood\_solutions()**

```
static std::vector< Solution > pmnk::Solution::neighborhood_solutions (
    RMNKEval & eval,
    Solution const & original ) [inline], [static]
```

Calculate all the neighbor solutions of the current one.

## Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>eval</i>      | The instance evaluator object.     |
| <i>generator</i> | The random number generator object |

## Returns

std::vector<Solution> A vector of solutions containing the current solution neighbor solutions.

**4.11.3.5 objective\_vector()**

```
ObjectiveVector const & pmnk::Solution::objective_vector ( ) const [inline]
```

Getter for the solution's objective vector.

**Returns**

ObjectiveVector const& A Read-Only reference to the solution's objective vector.

**4.11.3.6 operator=() [1/2]**

```
Solution & pmnk::Solution::operator= (
    Solution && other ) [default]
```

Defaulted move assignment operator.

**Parameters**

|              |   |
|--------------|---|
| <i>other</i> | A rvalue reference to solution to be copied |
|--------------|---|

**Returns**

[Solution](#)& A refernce to the solution to be assigned.

**4.11.3.7 operator=() [2/2]**

```
Solution & pmnk::Solution::operator= (
    Solution const & other ) [default]
```

Defaulted copy assignment operator.

**Parameters**

|              |  |
|--------------|--|
| <i>other</i> | A const lvalue reference to a solution to be copied. |
|--------------|--|

**Returns**

[Solution](#)& A lvalue reference to the solution to be assigned.

**4.11.3.8 operator[]()**

```
DecisionVector::reference pmnk::Solution::operator[] (
    std::size_t const i ) [inline]
```

Array Indexing operator overload. This operator provides a way to access the i-th bit in the solution's decision vector implementation.

## Parameters

|          |  |
|----------|--|
| <i>i</i> | The index of the element to be accessed. |
|----------|--|

## Returns

DecisionVector::reference A lvalue reference to value contained in the accessed index.

## 4.11.3.9 random\_solution()

```
template<typename RNG >
static Solution pmnk::Solution::random_solution (
    RMNKEval & eval,
    RNG & generator ) [inline], [static]
```

Build and evaluate a new random solution object.

## Template Parameters

|            |  |
|------------|--|
| <i>RNG</i> | The type for the random number generator object. |
|------------|--|

## Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>eval</i>      | The instance evaluator object.     |
| <i>generator</i> | The random number generator object |

## Returns

[Solution](#) A new [Solution](#) object containing a random solution.

## 4.11.3.10 size()

```
std::size_t pmnk::Solution::size ( ) const [inline], [noexcept]
```

Getter for the solution's decision/objective vector size.

## Returns

std::size\_t The size of this solution's decision/objective vector.

#### 4.11.3.11 uniform\_bit\_flip\_solution()

```
template<typename RNG >
static Solution pmnk::Solution::uniform_bit_flip_solution (
    RMNKEval & eval,
    RNG & generator,
    Solution const & original ) [inline], [static]
```

Build and evaluate a new Random solution object that results from a mutation in another solution's decision vector bit representation.

##### Template Parameters

|            |  |
|------------|--|
| <i>RNG</i> | The type for the random number generator object. |
|------------|--|

##### Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>eval</i>      | The instance evaluator object.     |
| <i>generator</i> | The random number generator object |
| <i>original</i>  | The parent solution to be mutated. |

##### Returns

[Solution](#) A new [Solution](#) object containing a random solution.

### 4.11.4 Friends And Related Function Documentation

#### 4.11.4.1 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    Solution const & solution ) [friend]
```

Extraction operator overload for this object.

##### Parameters

|                 |   |
|-----------------|---|
| <i>os</i>       | The output stream to where the solution string representation will be redirected. |
| <i>solution</i> | The solution whose representation will be inserted in the output stream           |

##### Returns

std::ostream& A output stream object with the solution's data appended to it.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/[solution.hpp](#)

## 4.12 pmnk::UniformCrossover< RNG > Struct Template Reference

Uniform Crossover operator.

```
#include <functor.hpp>
```

### Public Member Functions

- constexpr [UniformCrossover](#) (double crossover\_probability, RNG &rng)  
*Construct a new [UniformCrossover](#) object.*
- template<typename S = GASolution>  
constexpr void [operator\(\)](#) (S &s1, S &s2) noexcept  
*Function call operator overload. Implements the crossover operator functionality.*

### Public Attributes

- double **m\_crossover\_probability**
- RNG **m\_rng**
- std::bernoulli\_distribution **m\_distrib**

#### 4.12.1 Detailed Description

```
template<typename RNG>
struct pmnk::UniformCrossover< RNG >
```

Uniform Crossover operator.

#### Template Parameters

|            |  |
|------------|--|
| <i>RNG</i> | The type for the random number generator object. |
|------------|--|

#### 4.12.2 Constructor & Destructor Documentation

##### 4.12.2.1 UniformCrossover()

```
template<typename RNG >
constexpr pmnk::UniformCrossover< RNG >::UniformCrossover (
    double crossover_probability,
    RNG & rng ) [inline], [constexpr]
```

Construct a new [UniformCrossover](#) object.

## Parameters

|                              |  |
|------------------------------|--|
| <i>crossover_probability</i> | Crossover probability considered in this operator. |
| <i>rng</i>                   | The random number generator object instance.       |

### 4.12.3 Member Function Documentation

#### 4.12.3.1 operator()

```
template<typename RNG >
template<typename S = GASolution>
constexpr void pmnk::UniformCrossover< RNG >::operator() (
    S & s1,
    S & s2 ) [inline], [constexpr], [noexcept]
```

Function call operator overload. Implements the crossover operator functionality.

## Template Parameters

|   |   |
|---|---|
| S | The type used to store an genetic algorithm ( <a href="#">IBEA</a> ) solution |
|---|---|

## Parameters

|           |                              |
|-----------|------------------------------|
| <i>s1</i> | A solution to be recombined. |
| <i>s2</i> | A solution to be recombined. |

The documentation for this struct was generated from the following file:

- </home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp>

## 4.13 pmnk::UniformMutation< RNG > Struct Template Reference

Uniform Mutation operator.

```
#include <functor.hpp>
```

### Public Member Functions

- constexpr [UniformMutation](#) (double const mutation\_probability, RNG &rng)  
Construct a new [UniformMutation](#) object.
- template<typename S = GASolution>  
constexpr void [operator\(\)](#) (S &s) noexcept  
Function call operator overload. Implements the crossover operator functionality.



## Public Attributes

- double **m\_mutation\_probability**
- RNG **m\_rng**
- std::uniform\_real\_distribution< double > **m\_distrib**

### 4.13.1 Detailed Description

```
template<typename RNG>
struct pmnk::UniformMutation< RNG >
```

Uniform Mutation operator.

#### Template Parameters

|            |  |
|------------|--|
| <i>RNG</i> | The type for the random number generator object. |
|------------|--|

### 4.13.2 Constructor & Destructor Documentation

#### 4.13.2.1 UniformMutation()

```
template<typename RNG >
constexpr pmnk::UniformMutation< RNG >::UniformMutation (
    double const mutation_probability,
    RNG & rng ) [inline], [constexpr]
```

Construct a new [UniformMutation](#) object.

#### Parameters

|                             |   |
|-----------------------------|---|
| <i>mutation_probability</i> | Mutation probability considered in this operator. |
| <i>rng</i>                  | The random number generator object instance.      |

### 4.13.3 Member Function Documentation

#### 4.13.3.1 operator>()

```
template<typename RNG >
template<typename S = GASolution>
constexpr void pmnk::UniformMutation< RNG >::operator() (
    S & s ) [inline], [constexpr], [noexcept]
```

Function call operator overload. Implements the crossover operator functionality.

**Template Parameters**

|          |   |
|----------|---|
| <b>S</b> | The type used to store an genetic algorithm ( <a href="#">IBEA</a> ) solution |
|----------|---|

**Parameters**

|          |                           |
|----------|---------------------------|
| <b>s</b> | A solution to be mutated. |
|----------|---------------------------|

The documentation for this struct was generated from the following file:

- </home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp>

## Chapter 5

# File Documentation

### 5.1 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/↵ GSEMO/gsemo.hpp File Reference

GSEMO (Global Simple Multi-objective Optimizer) algorithm implementation.

```
#include <algorithm>
#include <csignal>
#include <iomanip>
#include <random>
#include "../Utils/solution.hpp"
#include "../Utils/utils.hpp"
#include "../Utils/wfg.hpp"
```

#### Classes

- class `pmnk::GSEMO`  
*Wrapper class for `GSEMO`.*

#### 5.1.1 Detailed Description

GSEMO (Global Simple Multi-objective Optimizer) algorithm implementation.

##### Author

Pedro Rodrigues ( `pedror@student.dei.uc.pt` )  
Alexandre Jesus ( `ajesus@dei.uc.pt` )

##### Version

0.1.0

##### Date

13-09-2021

##### Copyright

Copyright (c) 2021

## 5.2 gsemo.hpp

[Go to the documentation of this file.](#)

```

1
13 #ifndef GSEMO_HPP
14 #define GSEMO_HPP
15
16 #include <algorithm>
17 #include <csignal>
18 #include <iomanip>
19 #include <random>
20
21 #include "../Utils/solution.hpp"
22 #include "../Utils/Utils.hpp"
23 #include "../Utils/wfg.hpp"
24
25 namespace pmnk {
26
27 class GSEMO {
28 public:
29     RMNKEval eval;
30
31 private:
32
33     std::mt19937 m_generator;
34     std::ostream &m_os;
35
36     hvobj<typename ObjectiveVector::value_type> m_hvo;
37     std::vector<Solution> m_solutions;
38
39 public:
40
41     template <typename Str = std::string, typename Ref = ObjectiveVector>
42     GSEMO(Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)
43         : eval(std::forward<Str>(instance).c_str())
44         , m_generator(seed)
45         , m_os(os)
46         , m_hvo(std::forward<Ref>(ref)) {}
47
48     template <typename Str = std::string>
49     GSEMO(Str &&instance, unsigned int const seed, std::ostream &os)
50         : eval(std::forward<Str>(instance).c_str())
51         , m_generator(seed)
52         , m_os(os)
53         , m_hvo(ObjectiveVector(eval.getM(), 0)) {}
54
55     template <typename Str = std::string, typename Ref = ObjectiveVector>
56     GSEMO(Str &&instance, unsigned int const seed, Ref &&ref)
57         : GSEMO(std::forward<Str>(instance), seed, std::cout, std::forward<Ref>(ref)) {}
58
59     template <typename Str = std::string>
60     GSEMO(Str &&instance, unsigned int const seed)
61         : GSEMO(std::forward<Str>(instance), seed, std::cout) {}
62
63     template <typename Str = std::string, typename Ref = ObjectiveVector>
64     explicit GSEMO(Str &&instance, Ref &&ref)
65         : GSEMO(std::forward<Str>(instance), std::random_device()(), std::cout, std::forward<Ref>(ref)) {}
66
67     template <typename Str = std::string>
68     explicit GSEMO(Str &&instance)
69         : GSEMO(std::forward<Str>(instance), std::random_device()(), std::cout) {}
70
71     std::vector<Solution> const &solutions() const {
72         return m_solutions;
73     }
74
75     void run(std::size_t maxeval) {
76         auto rand_solution = Solution::random_solution(eval, m_generator);
77         m_hvo.insert(rand_solution.objective_vector());
78
79         add_non_dominated(m_solutions, std::move(rand_solution));
80         m_os << "evaluation, hypervolume\n";
81         m_os << std::setprecision(12) << 0 << ", " << m_hvo.value() << "\n";
82
83         for (std::size_t i = 0; i < maxeval; ++i) {
84             std::uniform_int_distribution<std::size_t> randint(0, m_solutions.size() - 1);
85
86             std::size_t index = randint(m_generator);
87             auto solution = Solution::uniform_bit_flip_solution(eval, m_generator, m_solutions[index]);
88
89             auto sov = solution.objective_vector();
90             if (add_non_dominated(m_solutions, std::move(solution))) {

```

```

162         m_hvo.insert(sov);
163         m_os « std::setprecision(12) « i + 1 « ", " « m_hvo.value() « "\n";
164     }
165 }
166 }
167 };
168 } // namespace pmnk
169 #endif //GSEMO_HPP

```

## 5.3 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp File Reference

Implementation of IBEA operators (using functors)

```

#include <iostream>
#include <random>
#include "../Utils/solution.hpp"
#include "../Utils/wfg.hpp"

```

### Classes

- struct `pmnk::IHD< R >`  
*Hypervolume Based IBEA indicator.*
- struct `pmnk::EPS`  
*Additive epsilon indicator.*
- struct `pmnk::NPointCrossover< RNG >`  
*N-Point crossover operator.*
- struct `pmnk::UniformCrossover< RNG >`  
*Uniform Crossover operator.*
- struct `pmnk::UniformMutation< RNG >`  
*Uniform Mutation operator.*
- struct `pmnk::KWayTournamentSelection< RNG >`  
*KWayTournamentSelection operator.*

### 5.3.1 Detailed Description

Implementation of IBEA operators (using functors)

#### Author

Pedro Rodrigues ( [pedror@student.dei.uc.pt](mailto:pedror@student.dei.uc.pt) )  
 Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

#### Version

0.1

#### Date

13-09-2021

#### Copyright

Copyright (c) 2021

## 5.4 functor.hpp

[Go to the documentation of this file.](#)

```

1
13 #ifndef IBEA_FUNCTOR_HPP
14 #define IBEA_FUNCTOR_HPP
15
16 #include <iostream>
17 #include <random>
18
19 #include "../Utils/solution.hpp"
20 #include "../Utils/wfg.hpp"
21
22 namespace pmnk {
23
24 template <typename R = ObjectiveVector>
25 struct IHD {
26     R m_ref;
27
28     constexpr explicit IHD(R &&ref)
29         : m_ref(std::forward<R>(ref)) {}
30
31     template <typename S = GASolution>
32     [[nodiscard]] double operator()(S const &s1, S const &s2) const {
33         auto const &o1 = s1.objective_vector();
34         auto const &o2 = s2.objective_vector();
35
36         if (weakly_dominates(o1, o2)) {
37             return point_hv(o2, m_ref) - point_hv(o1, m_ref);
38         } else {
39             hvobj<typename R::value_type> hvo(m_ref);
40             hvo.insert(o1), hvo.insert(o2);
41             return hvo.value() - point_hv(o1, m_ref);
42         }
43     }
44 };
45
46 struct EPS {
47     constexpr explicit EPS() = default;
48
49     template <typename S = GASolution>
50     [[nodiscard]] constexpr double operator()(S const &s1, S const &s2) const noexcept {
51         double indicator = std::numeric_limits<double>::min();
52         for (std::size_t i = 0; i < s1.objective_vector().size(); ++i) {
53             double const o1 = s1.objective_vector()[i];
54             double const o2 = s2.objective_vector()[i];
55             indicator = std::max(indicator, o2 - o1);
56         }
57         return indicator;
58     }
59 };
60
61 template <typename RNG>
62 struct NPointCrossover {
63     std::size_t m_crossover_points;
64     double m_crossover_probability;
65     RNG m_rng;
66     std::uniform_real_distribution<double> m_distrib;
67
68     constexpr NPointCrossover(std::size_t const crossover_points, double const crossover_probability,
69                               RNG &rng)
70         : m_crossover_points(crossover_points)
71         , m_crossover_probability(crossover_probability)
72         , m_rng(rng)
73         , m_distrib(0.0, 1.0) {}
74
75     template<typename S = GASolution>
76     void operator()(S &s1, S &s2) noexcept {
77         if (m_distrib(m_rng) < m_crossover_probability) {
78             std::size_t p1 = 0, p2 = 0;
79             for (std::size_t i = 0; i < m_crossover_points; ++i, p1 = p2) {
80                 std::uniform_int_distribution<std::size_t> randint(p1, s1.size() - 1);
81                 p2 = randint(m_rng);
82                 for (auto i = p1; i < p2; ++i) {
83                     std::swap(s1[i], s2[i]);
84                 }
85             }
86         }
87     }
88 };
89
90 template <typename RNG>

```

```

154 struct UniformCrossover {
155     double m_crossover_probability;
156     RNG m_rng;
157     std::bernoulli_distribution m_distrib;
158
159     constexpr UniformCrossover(double crossover_probability, RNG &rng)
160         : m_crossover_probability(crossover_probability)
161         , m_rng(rng)
162         , m_distrib() {}
163
164     template <typename S = GASolution>
165     constexpr void operator()(S &s1, S &s2) noexcept {
166         for (std::size_t i = 0; i < s1.size(); ++i) {
167             if (m_distrib(m_rng)) {
168                 std::swap(s1[i], s2[i]);
169             }
170         }
171     }
172 };
173
174 template <typename RNG>
175 struct UniformMutation {
176     double m_mutation_probability;
177     RNG m_rng;
178     std::uniform_real_distribution<double> m_distrib;
179
180     constexpr UniformMutation(double const mutation_probability, RNG &rng)
181         : m_mutation_probability(mutation_probability)
182         , m_rng(rng)
183         , m_distrib(0.0, 1.0) {}
184
185     template <typename S = GASolution>
186     constexpr void operator()(S &s) noexcept {
187         for (std::size_t i = 0; i < s.size(); ++i) {
188             if (m_distrib(m_rng) < m_mutation_probability) {
189                 s[i] = !s[i];
190             }
191         }
192     }
193 };
194
195 template <typename RNG>
196 struct KWayTournamentSelection {
197     std::size_t m_tournament_size;
198     std::size_t m_matting_pool_size;
199     RNG m_rng;
200
201     constexpr KWayTournamentSelection(std::size_t const tournament_size,
202                                       std::size_t const matting_pool_size, RNG &rng)
203         : m_tournament_size(tournament_size)
204         , m_matting_pool_size(matting_pool_size)
205         , m_rng(rng) {}
206
207     template <typename S = GASolution>
208     [[nodiscard]] std::vector<S> operator()(std::vector<S> const &population) noexcept {
209         std::vector<S> matting_pool;
210         matting_pool.reserve(m_matting_pool_size);
211         std::uniform_int_distribution<std::size_t> distrib(0, population.size() - 1);
212
213         for (std::size_t i = 0; i < m_matting_pool_size; ++i) {
214             auto best = distrib(m_rng);
215             for (std::size_t j = 0; j < m_tournament_size - 1; ++j) {
216                 auto other = distrib(m_rng);
217                 best = population[other].fitness() > population[best].fitness() ? other : best;
218             }
219             matting_pool.push_back(population[best]);
220         }
221         return matting_pool;
222     }
223 };
224
225 } // namespace pmnk
226 #endif // IBEA_FUNCTOR_HPP

```

## 5.5 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ibea.hpp File Reference

IBEA (Indicator Based Evolutionary Algorithm) implementation.

```
#include <csignal>
#include <iomanip>
#include <random>
#include "../Utils/solution.hpp"
#include "../Utils/Utils.hpp"
#include "../Utils/wfg.hpp"
#include "functor.hpp"
```

## Classes

- class `pmnk::IBEA`  
*Wrapper class for `IBEA`.*

### 5.5.1 Detailed Description

IBEA (Indicator Based Evolutionary Algorithm) implementation.

#### Author

Pedro Rodrigues ( [pedror@student.dei.uc.pt](mailto:pedror@student.dei.uc.pt) )  
Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

#### Version

0.1.0

#### Date

13-09-2021

#### Copyright

Copyright (c) 2021

## 5.6 ibea.hpp

[Go to the documentation of this file.](#)

```
1
12 #ifndef IBEA_HPP
13 #define IBEA_HPP
14
15 #include <csignal>
16 #include <iomanip>
17 #include <random>
18
19 #include "../Utils/solution.hpp"
20 #include "../Utils/Utils.hpp"
21 #include "../Utils/wfg.hpp"
22 #include "functor.hpp"
23
24 namespace pmnk {
25
26     class IBEA {
27     public:
28         RMNKEval eval;
```



```

30
31 private:
32     std::mt19937 m_generator;
33     std::ostream &m_os;
34
35     hvobj<ObjectiveVector::value_type> m_hvo;
36     std::vector<GASolution> m_solutions;
37
38 public:
39     template <typename Str = std::string, typename Ref = ObjectiveVector>
40     IBEA(Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)
41         : eval(std::forward<Str>(instance).c_str())
42         , m_generator(seed)
43         , m_os(os)
44         , m_hvo(std::forward<Ref>(ref)) {}
45
46     template <typename Str = std::string>
47     IBEA(Str &&instance, unsigned int const seed, std::ostream &os)
48         : eval(std::forward<Str>(instance).c_str())
49         , m_generator(seed)
50         , m_os(os)
51         , m_hvo(ObjectiveVector(eval.getM(), 0.0)) {}
52
53     template <typename Str = std::string, typename Ref = ObjectiveVector>
54     IBEA(Str &&instance, unsigned int const seed, Ref &&ref)
55         : IBEA(std::forward<Str>(instance), seed, std::cout, std::forward<Ref>(ref)) {}
56
57     template <typename Str = std::string>
58     IBEA(Str &&instance, unsigned int const seed)
59         : IBEA(std::forward<Str>(instance), seed, std::cout) {}
60
61     template <typename Str = std::string, typename Ref = ObjectiveVector>
62     explicit IBEA(Str &&instance, Ref &&ref)
63         : IBEA(std::forward<Str>(instance), std::random_device()(), std::cout,
64             std::forward<Ref>(ref)) {}
65
66     template <typename Str = std::string>
67     explicit IBEA(Str &&instance)
68         : IBEA(std::forward<Str>(instance), std::random_device()(), std::cout) {}
69
70     std::vector<GASolution> const &solutions() const {
71         return m_solutions;
72     }
73
74     template <typename I, typename S, typename M, typename C>
75     void run(std::size_t const maxeval, std::size_t const population_max_size,
76             std::size_t const max_generations, double const scaling_factor, I &&indicator,
77             C &&crossover_method, M &&mutation_method, S &&selection_method, bool adaptive) {
78         if (adaptive) {
79             m_run<true>(maxeval, population_max_size, max_generations, scaling_factor, indicator,
80                 crossover_method, mutation_method, selection_method);
81         } else {
82             m_run<false>(maxeval, population_max_size, max_generations, scaling_factor, indicator,
83                 crossover_method, mutation_method, selection_method);
84         }
85     }
86
87 private:
88     template <bool Adaptive, typename I, typename S, typename M, typename C>
89     void m_run(std::size_t const maxeval, std::size_t const pop_max,
90             std::size_t const max_generations, double const scaling_factor, I &&indicator,
91             C &&crossover_method, M &&mutation_method, S &&selection_method) {
92         std::size_t evaluation = 0, gen = 0;
93         double c = 1;
94
95         std::vector<GASolution> population;
96         population.reserve(pop_max);
97
98         std::cout << "evaluation,generation,hypervolume\n";
99
100         for (std::size_t i = 0; i < pop_max && evaluation < maxeval; ++i) {
101             auto sol = GASolution(Solution::random_solution(eval, m_generator));
102             if (add_non_dominated(m_solutions, sol)) {
103                 m_hvo.insert(sol.objective_vector());
104                 m_os << std::setprecision(12) << evaluation << "," << gen << "," << m_hvo.value() << "\n";
105             }
106             population.push_back(std::move(sol));
107             ++evaluation;
108         }
109
110         if (evaluation < maxeval) {
111             if constexpr (Adaptive) {
112                 c = m_adaptive_factor(population, indicator);
113             }
114             m_fitness_assignment(population, scaling_factor * c, indicator);
115         }
116     }

```

```

216     for (; evaluation < maxeval && gen < max_generations; ++gen) {
217         auto matting_pool = selection_method(population);
218
219         for (std::size_t i = 0; i < matting_pool.size() - 1; i += 2) {
220             crossover_method(matting_pool[i], matting_pool[i + 1]);
221         }
222
223         for (auto &individual : matting_pool) {
224             mutation_method(individual);
225             individual.eval(eval);
226         }
227
228         if constexpr (Adaptive) {
229             c = m_adaptive_factor(population, indicator);
230         }
231         m_fitness_assignment(population, scaling_factor * c, indicator);
232
233         for (auto &individual : matting_pool) {
234             if (add_non_dominated(m_solutions, individual)) {
235                 m_hvo.insert(individual.objective_vector());
236                 m_os << std::setprecision(12) << evaluation << ", " << gen << ", " << m_hvo.value() << "\n";
237             }
238             population.push_back(std::move(individual));
239             ++evaluation;
240         }
241         m_environmental_selection(population, scaling_factor * c, pop_max, indicator);
242     }
243     m_os << std::setprecision(12) << evaluation << ", " << gen << ", " << m_hvo.value() << "\n";
244 }
245
246 template <typename S = GASolution>
247 auto m_objective_bounds(std::vector<S> const &population) {
248     auto ub = std::numeric_limits<ObjectiveVector::value_type>::min();
249     auto lb = std::numeric_limits<ObjectiveVector::value_type>::max();
250     for (auto const &individual : population) {
251         for (auto const v : individual.objective_vector()) {
252             lb = std::min(lb, v);
253             ub = std::max(ub, v);
254         }
255     }
256     return std::make_pair(lb, ub);
257 }
258
259 template <typename S = GASolution>
260 auto m_scale_objective_vectors(std::vector<S> const &population, double const lb,
261                               double const ub) {
262     std::vector<S> s = population;
263     for (auto &individual : s) {
264         auto ov = individual.objective_vector();
265         for (auto &i : ov) {
266             i = (i - ub) / (ub - lb);
267         }
268         individual.set_objv(std::move(ov));
269     }
270     return s;
271 }
272
273 template <typename I, typename S = GASolution>
274 auto m_adaptive_factor(std::vector<S> const &population, I &&indicator) {
275     auto &[lb, ub] = m_objective_bounds(population);
276     auto s = m_scale_objective_vectors(population, lb, ub);
277     auto c = std::numeric_limits<ObjectiveVector::value_type>::min();
278     for (std::size_t i = 0; i < s.size(); ++i) {
279         for (std::size_t j = 0; j < s.size(); ++j) {
280             if (i != j) {
281                 c = std::max(c, std::abs(indicator(s[i], s[j])));
282             }
283         }
284     }
285     return c;
286 }
287
288 template <typename I, typename S = GASolution>
289 void m_fitness_assignment(std::vector<S> &population, double const k, I &&indicator) const {
290     for (std::size_t i = 0; i < population.size(); ++i) {
291         population[i].set_fitness(0);
292         for (std::size_t j = 0; j < population.size(); ++j) {
293             if (i != j) {
294                 population[i].set_fitness(population[i].fitness() -
295                                           std::exp(-indicator(population[j], population[i]) / k));
296             }
297         }
298     }
299 }
300
301 template <typename I, typename S = GASolution>
302 void m_environmental_selection(std::vector<S> &population, double const k,
303                               I &&indicator) {
304     auto &[lb, ub] = m_objective_bounds(population);
305     auto s = m_scale_objective_vectors(population, lb, ub);
306     auto c = std::numeric_limits<ObjectiveVector::value_type>::min();
307     for (std::size_t i = 0; i < s.size(); ++i) {
308         for (std::size_t j = 0; j < s.size(); ++j) {
309             if (i != j) {
310                 c = std::max(c, std::abs(indicator(s[i], s[j])));
311             }
312         }
313     }
314     return c;
315 }
316
317 template <typename I, typename S = GASolution>
318 void m_environmental_selection(std::vector<S> &population, double const k,
319                               I &&indicator) {
320     auto &[lb, ub] = m_objective_bounds(population);
321     auto s = m_scale_objective_vectors(population, lb, ub);
322     auto c = std::numeric_limits<ObjectiveVector::value_type>::min();
323     for (std::size_t i = 0; i < s.size(); ++i) {
324         for (std::size_t j = 0; j < s.size(); ++j) {
325             if (i != j) {
326                 c = std::max(c, std::abs(indicator(s[i], s[j])));
327             }
328         }
329     }
330     return c;
331 }
332
333 template <typename I, typename S = GASolution>
334 void m_environmental_selection(std::vector<S> &population, double const k,
335                               I &&indicator) {
336     auto &[lb, ub] = m_objective_bounds(population);
337     auto s = m_scale_objective_vectors(population, lb, ub);
338     auto c = std::numeric_limits<ObjectiveVector::value_type>::min();
339     for (std::size_t i = 0; i < s.size(); ++i) {
340         for (std::size_t j = 0; j < s.size(); ++j) {
341             if (i != j) {
342                 c = std::max(c, std::abs(indicator(s[i], s[j])));
343             }
344         }
345     }
346     return c;
347 }

```

```

351         std::size_t population_max_size, I &&indicator) {
352     while (population.size() > population_max_size) {
353         std::size_t worst = 0;
354         for (std::size_t i = 0; i < population.size(); ++i) {
355             worst = population[i].fitness() < population[worst].fitness() ? i : worst;
356         }
357         std::swap(population[worst], population.back());
358         for (std::size_t i = 0; i < population.size() - 1; ++i) {
359             population[i].set_fitness(population[i].fitness() +
360                                     std::exp(-indicator(population.back(), population[i]) / k));
361         }
362         population.pop_back();
363     }
364 }
365 }
366 };
367 } // namespace pmnk
368 #endif // IBEA_HPP

```

## 5.7 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/main.cpp File Reference

Driver program to test the implementation of some search algorithms and simplify the gathering of anytime data relevant to the study of their performance in the context of the pmnk-landscapes problem.

```

#include <CLI/CLI.hpp>
#include <fstream>
#include <iostream>
#include <random>
#include <tuple>
#include "GSEMO/gsemo.hpp"
#include "IBEA/ibea.hpp"
#include "PLS/pls.hpp"

```

### Macros

- #define [SELECTION](#)(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define [MUTATION](#)(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define [CROSSOVER](#)(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define [INDICATOR](#)(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define [RUN\\_IBEA\\_LOOP](#)(...) INDICATOR(\_\_VA\_ARGS\_\_)

### Enumerations

- enum class [Indicator](#) { [EPS](#) , [IHD](#) }  
*Enum representing which indicator to use as the IBEA algorithm indicator operator. Possible values are:*
- enum class [Crossover](#) { [NPC](#) , [UC](#) }  
*Enum representing which indicator to use as the IBEA algorithm crossover operator. Possible values are:*
- enum class [Selection](#) { [KWT](#) }  
*Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:*
- enum class [Mutation](#) { [UM](#) }  
*Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:*

## Functions

- void [set\\_positional\\_arguments](#) (CLI::App &app, std::string &instance)  
*Set the CLI positional arguments options/flags.*
- void [set\\_general\\_options](#) (CLI::App &app, std::size\_t &maxeval, unsigned int &seed, std::string &output, ObjectiveVector &ref)  
*Set the CLI general options/flags.*
- void [set\\_pls\\_options](#) (CLI::App &app, PLSAcceptanceCriterion &pac, PLSExplorationCriterion &pne)  
*Set the PLS algorithm options/flags.*
- void [set\\_ibeal\\_options](#) (CLI::App &app, std::size\_t &population\_size, std::size\_t &generations, double &scaling\_factor, bool &adaptive)  
*Set the IBEA algorithm options/flags.*
- void [set\\_ibeal\\_mutation\\_options](#) (CLI::App &app, double &mutation\_probability)  
*Set the IBEA algorithm mutation operators options/flags.*
- void [set\\_ibeal\\_crossover\\_options](#) (CLI::App &app, double &crossover\_probability, std::size\_t &npoints)  
*Set the IBEA algorithm crossover operators options/flags.*
- void [set\\_ibeal\\_selection\\_options](#) (CLI::App &app, std::size\_t &matting\_pool\_size, std::size\_t &tournament\_size)  
*Set the ibea selection options object.*
- void [gsemo](#) (std::string const &instance, std::size\_t const maxeval, unsigned int const seed, std::ostream &os, ObjectiveVector &ref)  
*CLI::App callback for the gsemo algorithm.*
- void [pls](#) (std::string const &instance, std::size\_t maxeval, unsigned int seed, PLSAcceptanceCriterion const pac, PLSExplorationCriterion const pne, std::ostream &os, ObjectiveVector &ref)  
*CLI::App callback for the pls algorithm.*
- void [ibeal](#) (std::string const &instance, std::size\_t const maxeval, unsigned int const seed, std::size\_t const ps, std::size\_t const gen, double const k, double const mp, double const cp, std::size\_t npts, std::size\_t const mps, std::size\_t const ts, [Indicator](#) const indicator, [Crossover](#) const crossover, [Mutation](#) const mutation, [Selection](#) const selection, bool adaptive, std::ostream &os, ObjectiveVector &ref)  
*CLI::App callback for the ibea algorithm.*
- int **main** (int argc, char \*\*argv)

### 5.7.1 Detailed Description

Driver program to test the implementation of some search algorithms and simplify the gathering of anytime data relevant to the study of their performance in the context of the pmnk-landscapes problem.

#### Author

Pedro Rodrigues ( [pedror@student.dei.uc.pt](mailto:pedror@student.dei.uc.pt) )  
 Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

#### Version

0.1.0

#### Date

13-09-2021

#### Copyright

Copyright (c) 2021

## 5.7.2 Macro Definition Documentation

### 5.7.2.1 CROSSOVER

```
#define CROSSOVER(
    MAXEVAL,
    POP,
    GEN,
    FACTOR,
    I,
    C,
    M,
    S,
    ADAPT )
```

**Value:**

```
switch (C) {
case Crossover::NPC:
    MUTATION(MAXEVAL, POP, GEN, FACTOR, I, NPointCrossover(npts, cp, rng), M, S, ADAPT)
    break;
case Crossover::UC:
    MUTATION(MAXEVAL, POP, GEN, FACTOR, I, UniformCrossover(cp, rng), M, S, ADAPT)
    break;
default:
    throw("Unknown crossover operator!\n");
}
```

### 5.7.2.2 INDICATOR

```
#define INDICATOR(
    MAXEVAL,
    POP,
    GEN,
    FACTOR,
    I,
    C,
    M,
    S,
    ADAPT )
```

**Value:**

```
switch (I) {
case Indicator::EPS:
    CROSSOVER(MAXEVAL, POP, GEN, FACTOR, EPS(), C, M, S, ADAPT)
    break;
case Indicator::IHD:
    CROSSOVER(MAXEVAL, POP, GEN, FACTOR, IHD(ObjectiveVector(ibea.eval.getM(), 0)), C, M, S,
    ADAPT)
    break;
default:
    throw("Unknown indicator!\n");
}
```

### 5.7.2.3 MUTATION

```
#define MUTATION(
    MAXEVAL,
    POP,
    GEN,
    FACTOR,
    I,
    C,
    M,
    S,
    ADAPT )
```

#### Value:

```
switch (M) {
case Mutation::UM:
    SELECTION(MAXEVAL, POP, GEN, FACTOR, I, C, UniformMutation(mp, rng), S, ADAPT)
    break;
default:
    throw("Unknown mutation operator!\n");
}
```

### 5.7.2.4 SELECTION

```
#define SELECTION(
    MAXEVAL,
    POP,
    GEN,
    FACTOR,
    I,
    C,
    M,
    S,
    ADAPT )
```

#### Value:

```
switch (S) {
case Selection::KWT:
    if (ref.empty()) {
        IBEA ibea(instance, seed, os);
        ibea.run(MAXEVAL, POP, GEN, FACTOR, I, C, M, KWayTournamentSelection(ts, mps, rng),
            ADAPT);
    } else {
        IBEA ibea(instance, seed, os, ref);
        ibea.run(MAXEVAL, POP, GEN, FACTOR, I, C, M, KWayTournamentSelection(ts, mps, rng),
            ADAPT);
    }
    break;
default:
    throw("Unknown selection method!");
}
```

## 5.7.3 Enumeration Type Documentation

### 5.7.3.1 Crossover

```
enum class Crossover [strong]
```

Enum representing which indicator to use as the IBEA algorithm crossover operator. Possible values are:

- NPC (N-Point Crossover)
- UC (Uniform Crossover)

### 5.7.3.2 Indicator

```
enum class Indicator [strong]
```

Enum representing which indicator to use as the IBEA algorithm indicator operator. Possible values are:

- EPS (Additive Epsilon Indicator)
- IHD (Hypervolume Based Indicator)

### 5.7.3.3 Mutation

```
enum class Mutation [strong]
```

Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:

- UM (Uniform Mutation)

### 5.7.3.4 Selection

```
enum class Selection [strong]
```

Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:

- KWT (K-Way Tournament Selection)

## 5.7.4 Function Documentation

### 5.7.4.1 gsemo()

```
void gsemo (  
    std::string const & instance,  
    std::size_t const maxeval,  
    unsigned int const seed,  
    std::ostream & os,  
    ObjectiveVector & ref ) [inline]
```

CLI::App callback for the gsemo algorithm.

## Parameters

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>maxeval</i>  | The maximum number of evaluations performed by the algorithms (stopping criterion)                                     |
| <i>seed</i>     | The seed used by the pseudo random number generator used in these algorithms   |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

## 5.7.4.2 ibea()

```
void ibea (
    std::string const & instance,
    std::size_t const maxeval,
    unsigned int const seed,
    std::size_t const ps,
    std::size_t const gen,
    double const k,
    double const mp,
    double const cp,
    std::size_t npts,
    std::size_t mps,
    std::size_t ts,
    Indicator const indicator,
    Crossover const crossover,
    Mutation const mutation,
    Selection const selection,
    bool adaptive,
    std::ostream & os,
    ObjectiveVector & ref ) [inline]
```

CLI::App callback for the ibea algorithm.

## Parameters

|                  |  |
|------------------|--|
| <i>instance</i>  | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>maxeval</i>   | The maximum number of evaluations performed by the algorithms (stopping criterion)                                     |
| <i>seed</i>      | The seed used by the pseudo random number generator used in these algorithms   |
| <i>ps</i>        | The maximum population size  |
| <i>gen</i>       | The maximum number of generations  |
| <i>k</i>         | The scaling factor   |
| <i>mp</i>        | The mutation probability considered by the mutation operator   |
| <i>cp</i>        | The crossover probability considered by the crossover operator   |
| <i>npts</i>      | The number of crossover points considered by the NPointCrossover operator if used                                      |
| <i>mps</i>       | The matting pool size  |
| <i>ts</i>        | The size of the tournament considered by the KWayTournament operator   |
| <i>indicator</i> | The indicator to be used by the IBEA indicator operator  |
| <i>crossover</i> | The crossover method considered by the IBEA crossover operator   |



**Parameters**

|                  |  |
|------------------|--|
| <i>mutation</i>  | The mutation method considered by the IBEA mutation operator   |
| <i>selection</i> | The selection method considered by the IBEA selection operator   |
| <i>adaptive</i>  | boolean indicative of version of IBEA to be used. If true use adaptive version of (A-IBEA) else use (B-IBEA) |
| <i>os</i>        | The name of the output file where the standard output stream should be redirected                            |
| <i>ref</i>       | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)      |

Helper define to avoid the use of runtime polymorphism methods to distinguish between selection operators that ibea is going to use during its execution

Helper define to avoid the use of runtime polymorphism methods to distinguish between crossover operators that ibea is going to use during its execution

Helper define to avoid the use of runtime polymorphism methods to distinguish between crossover operators that ibea is going to use during its execution.

\ Helper define to avoid the use of runtime polymorphism methods to distinguish between indicator operators that ibea is going to use during its execution.

Helper define that forwards all the information to the others

**5.7.4.3 pls()**

```
void pls (
    std::string const & instance,
    std::size_t maxeval,
    unsigned int seed,
    PLSAcceptanceCriterion const pac,
    PLSExplorationCriterion const pne,
    std::ostream & os,
    ObjectiveVector & ref ) [inline]
```

CLI::App callback for the pls algorithm.

**Parameters**

|                 |  |
|-----------------|--|
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |
| <i>maxeval</i>  | The maximum number of evaluations performed by the algorithms (stopping criterion)                                     |
| <i>seed</i>     | The seed used by the pseudo random number generator used in these algorithms   |
| <i>pac</i>      | The PLS algorithm solution acceptance criterion  |
| <i>pne</i>      | The PLS algorithm neighborhood solutions exploration criterion   |
| <i>os</i>       | The name of the output file where the standard output stream should be redirected                                      |
| <i>ref</i>      | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)                |

#### 5.7.4.4 set\_general\_options()

```
void set_general_options (
    CLI::App & app,
    std::size_t & maxeval,
    unsigned int & seed,
    std::string & output,
    ObjectiveVector & ref ) [inline]
```

Set the CLI general options/flags.

##### Parameters

|                |   |
|----------------|---|
| <i>app</i>     | CLI::App object that will hold all the global options/flags (below).                                    |
| <i>maxeval</i> | The maximum number of evaluations performed by the algorithms (stopping criterion)                      |
| <i>seed</i>    | The seed used by the pseudo random number generator used in these algorithms                            |
| <i>output</i>  | The name of the output file where the standard output stream should be redirected                       |
| <i>ref</i>     | The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure) |

#### 5.7.4.5 set\_ibea\_crossover\_options()

```
void set_ibea_crossover_options (
    CLI::App & app,
    double & crossover_probability,
    std::size_t & npoints ) [inline]
```

Set the IBEA algorithm crossover operators options/flags.

##### Parameters

|                              |   |
|------------------------------|---|
| <i>app</i>                   | CLI::App object that will hold all the IBEA crossover operator options/flags (below).   |
| <i>crossover_probability</i> | The probability of occurrence of crossover between two individual's genotypes   |
| <i>npoints</i>               | If the crossover operator is the n-point crossover this option is set. This option represents the number of crossover points to be considered when swapping both individual's genetic content |

#### 5.7.4.6 set\_ibea\_mutation\_options()

```
void set_ibea_mutation_options (
    CLI::App & app,
    double & mutation_probability ) [inline]
```

Set the IBEA algorithm mutation operators options/flags.

## Parameters

|                             |  |
|-----------------------------|--|
| <i>app</i>                  | CLI::App object that will hold all the IBEA mutation operator options/flags (below). |
| <i>mutation_probability</i> | The probability of occurrence of mutations in the individual's genotype              |

**5.7.4.7 set\_ibea\_options()**

```
void set_ibea_options (
    CLI::App & app,
    std::size_t & population_size,
    std::size_t & generations,
    double & scaling_factor,
    bool & adaptive ) [inline]
```

Set the IBEA algorithm options/flags.

## Parameters

|                        |  |
|------------------------|--|
| <i>app</i>             | CLI::App object that will hold all the IBEA options/flags (below).   |
| <i>population_size</i> | The maximum size the population  |
| <i>generations</i>     | The maximum number of generations (stopping criterion)   |
| <i>scaling_factor</i>  | The IBEA scaling factor  |
| <i>adaptive</i>        | A boolean indicative of the version of the algorithm to be used. True for B-IBEA (Basic IBEA) and false for A-IBEA (Adaptive IBEA) |

**5.7.4.8 set\_ibea\_selection\_options()**

```
void set_ibea_selection_options (
    CLI::App & app,
    std::size_t & matting_pool_size,
    std::size_t & tournament_size ) [inline]
```

Set the ibea selection options object.

## Parameters

|                          |   |
|--------------------------|---|
| <i>app</i>               | CLI::App object that will hold all the IBEA selection operator options/flags (below). |
| <i>matting_pool_size</i> | The size of the matting pool.   |
| <i>tournament_size</i>   | The size of the tournament used for selection.  |

**5.7.4.9 set\_pls\_options()**

```
void set_pls_options (
```

```

CLI::App & app,
PLSAcceptanceCriterion & pac,
PLSExplorationCriterion & pne ) [inline]

```

Set the PLS algorithm options/flags.

#### Parameters

|            |   |
|------------|---|
| <i>app</i> | CLI::App object that will hold all the PLS options/flags (below). |
| <i>pac</i> | The PLS algorithm solution acceptance criterion                   |
| <i>pne</i> | The PLS algorithm neighborhood solutions exploration criterion    |

#### 5.7.4.10 set\_positional\_arguments()

```

void set_positional_arguments (
    CLI::App & app,
    std::string & instance ) [inline]

```

Set the CLI positional arguments options/flags.

#### Parameters

|                 |  |
|-----------------|--|
| <i>app</i>      | CLI::App object that will hold all the options/flags of the positional arguments (below).                              |
| <i>instance</i> | The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script |

## 5.8 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/↵ PLS/pls.hpp File Reference

PLS (Pareto Local Search) algorithm implementation.

```

#include <csignal>
#include <iomanip>
#include "../Utils/solution.hpp"
#include "../Utils/Utils.hpp"
#include "../Utils/wfg.hpp"

```

### Classes

- class [pmnk::PLS](#)  
Wrapper class for [PLS](#).

### Macros

- #define [RUNLOOP](#)(FIRSTIMPROV)

## Enumerations

- enum class **PLSAcceptanceCriterion** { **NON\_DOMINATING** , **DOMINATING** , **BOTH** }
- enum class **PLSExplorationCriterion** { **BEST\_IMPROVEMENT** , **FIRST\_IMPROVEMENT** , **BOTH** }

### 5.8.1 Detailed Description

PLS (Pareto Local Search) algorithm implementation.

#### Author

Pedro Rodrigues ( [pedror@student.dei.uc.pt](mailto:pedror@student.dei.uc.pt) )

Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

#### Version

0.1.0

#### Date

13-09-2021

#### Copyright

Copyright (c) 2021

### 5.8.2 Macro Definition Documentation

#### 5.8.2.1 RUNLOOP

```
#define RUNLOOP(  
    FIRSTIMPROV )
```

#### Value:

```
switch (acceptance_criterion) {  
    case PLSAcceptanceCriterion::NON_DOMINATING:  
        m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::NON_DOMINATING>(eval, maxeval);  
        break;  
    case PLSAcceptanceCriterion::DOMINATING:  
        m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::DOMINATING>(eval, maxeval);  
        break;  
    case PLSAcceptanceCriterion::BOTH:  
        m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::BOTH>(eval, maxeval);  
        break;  
    default:  
        throw("Unknown value for acceptance criterion");  
}
```

## 5.9 pls.hpp

[Go to the documentation of this file.](#)

```

1
13 #ifndef PLS_HPP
14 #define PLS_HPP
15
16 #include <csignal>
17 #include <iomanip>
18
19 #include "../Utils/solution.hpp"
20 #include "../Utils/Utils.hpp"
21 #include "../Utils/wfg.hpp"
22
23 namespace pmnk {
24
25 // Acceptance criterion:
26 // - 0 -> accept every non-dominated neighbor (NON_DOMINATING)
27 // - 1 -> accept only neighbors that dominate current solution (DOMINATING)
28 // - 2 -> first try to accept only neighbors that dominate the (BOTH)
29 //       current solution, if none exist accept non-dominated
30
31 enum class PLSAcceptanceCriterion { NON_DOMINATING, DOMINATING, BOTH };
32
33 // Neighborhood exploration:
34 // - 0 -> explore every acceptable neighbor (BEST_IMPROVEMENT)
35 // - 1 -> stop once one neighbor is accepted (FIRST_IMPROVEMENT)
36 // - 2 -> use 1 until PLS stops, afterwards restart and use 0 (BEST_IMPROVEMENT)
37 enum class PLSExplorationCriterion { BEST_IMPROVEMENT, FIRST_IMPROVEMENT, BOTH };
38
39 class PLS {
40 public:
41     RMNKEval eval;
42
43 private:
44     std::mt19937 m_generator;
45     std::ostream &m_os;
46
47     hvobj<typename ObjectiveVector::value_type> m_hvo;
48     std::vector<Solution> m_solutions;
49     std::vector<Solution> m_non_visited_solutions;
50
51 public:
52
53     template <typename Str = std::string, typename Ref = ObjectiveVector>
54     PLS(Str &&instance, unsigned int seed, std::ostream &os, Ref &&ref)
55         : eval(std::forward<Str>(instance).c_str())
56         , m_generator(seed)
57         , m_os(os)
58         , m_hvo(std::forward<Ref>(ref)) {}
59
60     template <typename Str = std::string>
61     PLS(Str &&instance, unsigned int seed, std::ostream &os)
62         : eval(std::forward<Str>(instance).c_str())
63         , m_generator(seed)
64         , m_os(os)
65         , m_hvo(ObjectiveVector(eval.getM(), 0.0)) {}
66
67     template <typename Str = std::string, typename Ref = ObjectiveVector>
68     PLS(Str &&instance, unsigned int seed, Ref &&ref)
69         : PLS(std::forward<Str>(instance), seed, std::cout, std::forward<Ref>(ref)) {}
70
71     template <typename Str = std::string>
72     PLS(Str &&instance, unsigned int seed)
73         : PLS(std::forward<Str>(instance), seed, std::cout) {}
74
75     template <typename Str = std::string, typename Ref = ObjectiveVector>
76     PLS(Str &&instance, Ref &&ref)
77         : PLS(std::forward<Str>(instance), std::random_device()(), std::cout,
78             std::forward<Ref>(ref)) {}
79
80     template <typename Str = std::string>
81     PLS(Str &&instance)
82         : PLS(std::forward<Str>(instance), std::random_device()(), std::cout) {}
83
84     std::vector<Solution> const solutions() const {
85         return m_solutions;
86     }
87
88     std::vector<Solution> const non_visited_solutions() const {
89         return m_non_visited_solutions;
90     }
91
92     void run(std::size_t maxeval, PLSAcceptanceCriterion const acceptance_criterion,
93             PLSExplorationCriterion const neighborhood_exploration) {

```

```

170     auto rand_solution = Solution::random_solution(eval, m_generator);
171     m_hvo.insert(rand_solution.objective_vector());
172
173     add_non_dominated(m_non_visited_solutions, std::move(rand_solution));
174     m_solutions = m_non_visited_solutions;
175
176     std::size_t eval = 0;
177     m_os << "evaluation, hypervolume\n";
178     m_os << std::setprecision(12) << eval << ", " << m_hvo.value() << "\n";
179
180 #define RUNLOOP(FIRSTIMPROV)
181     switch (acceptance_criterion) {
182     case PLSAcceptanceCriterion::NON_DOMINATING:
183         m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::NON_DOMINATING>(eval, maxeval);
184         break;
185     case PLSAcceptanceCriterion::DOMINATING:
186         m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::DOMINATING>(eval, maxeval);
187         break;
188     case PLSAcceptanceCriterion::BOTH:
189         m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::BOTH>(eval, maxeval);
190         break;
191     default:
192         throw("Unknown value for acceptance criterion");
193     }
194
195     if (neighborhood_exploration == PLSExplorationCriterion::BEST_IMPROVEMENT) {
196         RUNLOOP(false);
197     } else if (neighborhood_exploration == PLSExplorationCriterion::FIRST_IMPROVEMENT) {
198         RUNLOOP(true);
199     } else if (neighborhood_exploration == PLSExplorationCriterion::BOTH) {
200         RUNLOOP(true);
201         RUNLOOP(false);
202     } else {
203         throw("Unknown value for neighborhood exploration");
204     }
205 }
206
207 private:
208
209 template <bool FirstImprov, PLSAcceptanceCriterion Acceptance>
210 void m_loop(size_t &evaluation, size_t maxeval) {
211     while (evaluation < maxeval && !m_non_visited_solutions.empty()) {
212         std::uniform_int_distribution<std::size_t> distrib(0, m_non_visited_solutions.size() - 1);
213         std::size_t index = distrib(m_generator);
214
215         auto original = std::move(m_non_visited_solutions[index]);
216         m_non_visited_solutions[index] = std::move(m_non_visited_solutions.back());
217         m_non_visited_solutions.pop_back();
218
219         if constexpr (Acceptance == PLSAcceptanceCriterion::NON_DOMINATING) {
220             for (size_t i = 0; i < original.decision_vector().size() && evaluation < maxeval; ++i) {
221                 DecisionVector decision_vector = original.decision_vector();
222                 decision_vector[i] = !decision_vector[i];
223                 auto solution = Solution(eval, std::move(decision_vector));
224                 ++evaluation;
225                 if (add_non_dominated(m_solutions, solution)) {
226                     m_hvo.insert(solution.objective_vector());
227                     add_non_dominated(m_non_visited_solutions, std::move(solution));
228                     m_os << std::setprecision(12) << evaluation << ", " << m_hvo.value() << "\n";
229                     if constexpr (FirstImprov) {
230                         break;
231                     }
232                 }
233             }
234         } else if constexpr (Acceptance == PLSAcceptanceCriterion::DOMINATING) {
235             for (size_t i = 0; i < original.decision_vector().size() && evaluation < maxeval; ++i) {
236                 DecisionVector decision_vector = original.decision_vector();
237                 decision_vector[i] = !decision_vector[i];
238                 auto solution = Solution(eval, std::move(decision_vector));
239                 ++evaluation;
240                 if (solution.dominance(original) == DominanceType::DOMINATES &&
241                     add_non_dominated(m_solutions, solution)) {
242                     m_hvo.insert(solution.objective_vector());
243                     add_non_dominated(m_non_visited_solutions, std::move(solution));
244                     m_os << std::setprecision(12) << evaluation << ", " << m_hvo.value() << "\n";
245                     if constexpr (FirstImprov) {
246                         break;
247                     }
248                 }
249             }
250         } else if constexpr (Acceptance == PLSAcceptanceCriterion::BOTH) {
251             std::vector<std::pair<Solution, size_t>> remaining;
252             remaining.reserve(original.decision_vector().size());
253             bool use_remaining = true;
254             for (size_t i = 0; i < original.decision_vector().size() && evaluation < maxeval; ++i) {
255                 DecisionVector decision_vector = original.decision_vector();
256                 decision_vector[i] = !decision_vector[i];

```

```

268         auto solution = Solution(eval, std::move(decision_vector));
269         ++evaluation;
270         if (solution.dominance(original) == DominanceType::DOMINATES &&
271             add_non_dominated(m_solutions, solution)) {
272             use_remaining = false;
273             m_hvo.insert(solution.objective_vector());
274             add_non_dominated(m_non_visited_solutions, std::move(solution));
275             m_os << std::setprecision(12) << evaluation << "," << m_hvo.value() << "\n";
276             if constexpr (FirstImprov) {
277                 break;
278             }
279         } else if (use_remaining) {
280             remaining.emplace_back(std::move(solution), evaluation);
281         }
282     }
283     if (use_remaining) {
284         for (auto &&[solution, iteration] : remaining) {
285             if (add_non_dominated(m_solutions, solution)) {
286                 m_hvo.insert(solution.objective_vector());
287                 add_non_dominated(m_non_visited_solutions, std::move(solution));
288                 m_os << std::setprecision(12) << iteration << "," << m_hvo.value() << "\n";
289                 if constexpr (FirstImprov) {
290                     break;
291                 }
292             }
293         }
294     }
295 }
296 }
297 }
298 };
299 } // namespace pmnk
300 #endif // PLS_HPP

```

## 5.10 rMNKEval.hpp

```

1  /*
2   This library is free software; you can redistribute it and/or
3   modify it under the terms of the GNU Lesser General Public
4   License as published by the Free Software Foundation; version 3
5   of the License.
6
7   This library is distributed in the hope that it will be useful,
8   but WITHOUT ANY WARRANTY; without even the implied warranty of
9   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
10  Lesser General Public License for more details.
11
12  You should have received a copy of the GNU Lesser General Public
13  License along with this library; if not, write to the Free Software
14  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
15
16  Contact: http://mocobench.sourceforge.net
17
18  Authors:
19      Arnaud Liefioghe <arnaud.liefioghe@lifel.fr>
20      Sebastien Verel <sebastien.verel@inria.fr>
21  */
22
23  #ifndef __rMNKEval
24  #define __rMNKEval
25
26  /*
27   * Fitness function of the rMNK-landscapes
28   * reading a rhoMNK-landscapes instance file
29   * in c++ style
30   *
31   * rhoMNK-landscapes instances can be generated with the rmnkGenerator.R
32   *
33   * More information on rhoMNK-landscapes, see original paper:
34   * Verel S., Liefioghe A., Jourdan L., Dhaenens C. "Analyzing the Effect of Objective Correlation
35   * on the Efficient Set of MNK-Landscapes", In Proceedings of Learning and Intelligent OptimizatiON
36   * Conference (LION 5), LNCS, p. , 2011.
37   */
38
39  #include <fstream>
40  #include <iostream>
41  #include <string>
42  #include <vector>
43
44  namespace pmnk {
45
46  class RMNKEval {
47  public:

```



```

49  /*
50  * Constructor
51  *
52  * @param _fileName file name instance of the rho MNK-landscapes
53  */
54  RMNKEval(const char *_fileName) {
55      load(_fileName);
56  }
57
58  /*
59  * Destructor
60  */
61  ~RMNKEval() {
62      if (links != NULL) {
63          for (unsigned int n = 0; n < M; n++) {
64              for (unsigned int i = 0; i < N; i++)
65                  delete[] (links[n][i]);
66              delete[] links[n];
67          }
68
69          delete[] links;
70          links = NULL;
71      }
72
73      if (tables != NULL) {
74          for (unsigned int n = 0; n < M; n++) {
75              for (unsigned int i = 0; i < N; i++)
76                  delete[] (tables[n][i]);
77              delete[] tables[n];
78          }
79
80          delete tables;
81          tables = NULL;
82      }
83  }
84
85  /*
86  * Compute the fitness function
87  *
88  * @param _solution the solution to evaluate
89  * @param _objVec the objective vector of the corresponding solution
90  */
91  void eval(std::vector<bool> &_solution, std::vector<double> &_objVec) {
92      _objVec.resize(M);
93
94      for (unsigned n = 0; n < M; n++)
95          _objVec[n] = evalNK(n, _solution);
96  }
97
98  /*
99  * to get objective space dimension
100  *
101  * @return dimension of the objective space
102  */
103  unsigned getM() {
104      return M;
105  }
106
107  /*
108  * to get bitstring size
109  *
110  * @return dimension of the bitstring
111  */
112  unsigned getN() {
113      return N;
114  }
115
116  /*
117  * to get epistasis degree (K)
118  *
119  * @return epistasis degree K
120  */
121  unsigned getK() {
122      return K;
123  }
124
125  /*
126  * to get the correlation between each tuple of contributions
127  *
128  * @return parameter rho
129  */
130  double getRho() {
131      return rho;
132  }
133
134  protected:
135  // correlation between contributions

```

```

136 double rho;
137
138 // number of objective functions
139 unsigned M;
140
141 // size of the bit string
142 unsigned N;
143
144 // number of interactions between variables (epistasis)
145 unsigned K;
146
147 // the M tables of contributions
148 double ***tables;
149
150 // the M links description
151 unsigned ***links;
152
153 /*****
154  *
155  * Load the file of a rMNK-landscapes instance
156  *
157  * @param fileName file name instance of the rMNK-landscapes
158  *
159  *****/
160 virtual void load(const char *_fileName) {
161     std::fstream file;
162     file.open(_fileName, std::ios::in);
163
164     if (file.is_open()) {
165         std::string s;
166
167         // read the commentaries
168         std::string line;
169
170         file >> s;
171         while (s[0] == 'c') {
172             getline(file, line, '\n');
173             file >> s;
174         }
175
176         // read the parameters
177         if (s.compare("p") != 0)
178             std::cerr << "Error RMNKEval.load: expected line beging by \"p\" at " + s + " in "
179                 << _fileName;
180
181         file >> s;
182
183         if (s.compare("rMNK") != 0)
184             std::cerr << "Error RMNKEval.load: type rMNK expected at " + s + " in " << _fileName;
185
186         // effective read of the parameters
187         file >> rho >> M >> N >> K;
188
189         init();
190
191         file >> s;
192         // read the links
193         if (s.compare("p") != 0)
194             std::cerr << "Error RMNKEval.load: expected line beging by \"p\" at " + s + " in "
195                 << _fileName;
196
197         file >> s;
198         if (s.compare("links") == 0)
199             loadLinks(file);
200         else
201             std::cerr << "Error RMNKEval.load: line with \"links\" expected at " + s + " in "
202                 << _fileName;
203
204         // read the tables of contributions
205         file >> s;
206         if (s.compare("p") != 0)
207             std::cerr << "Error RMNKEval.load: expected line beging by \"p\" at " + s + " in "
208                 << _fileName;
209
210         file >> s;
211
212         if (s.compare("tables") == 0)
213             loadTables(file);
214         else
215             std::cerr << "Error RMNKEval.load: line with \"tables\" expected at " + s + " in "
216                 << _fileName;
217
218         file.close();
219     } else
220         std::cerr << "Error RMNKEval.load: impossible to open file " << _fileName;
221 };
222

```

```

223  /*****
224  *
225  * Initialization of the different tables and epistasis links
226  *
227  *****/
228  void init() {
229      links = new unsigned **[M];
230      tables = new double **[M];
231
232      for (unsigned n = 0; n < M; n++) {
233          links[n] = new unsigned *[N];
234          tables[n] = new double *[N];
235
236          for (unsigned i = 0; i < N; i++) {
237              tables[n][i] = new double[1 « (K + 1)];
238              links[n][i] = new unsigned[K + 1];
239          }
240      }
241  }
242
243  /*****
244  *
245  * Load the epistasis links from file
246  *
247  * @param _file open file of the instance
248  *
249  *****/
250  void loadLinks(std::fstream &_file) {
251      unsigned n, i, j;
252
253      for (i = 0; i < N; i++)
254          for (j = 0; j < K + 1; j++)
255              for (n = 0; n < M; n++)
256                  _file » links[n][i][j];
257  }
258
259  /*****
260  *
261  * Load the tables of contribution
262  *
263  * @param _file open file of the instance
264  *
265  *****/
266  void loadTables(std::fstream &_file) {
267      unsigned n, i;
268      int j;
269
270      for (i = 0; i < N; i++)
271          for (j = 0; j < (1 « (K + 1)); j++)
272              for (n = 0; n < M; n++)
273                  _file » tables[n][i][j];
274  }
275
276  /*****
277  *
278  * Fitness function of a single-objective NK-landscapes
279  *
280  * @param _numObj the objective function to consider
281  * @param _sol the solution to evaluate
282  *
283  *****/
284  double evalNK(unsigned _numObj, std::vector<bool> &_sol) {
285      double accu = 0.0;
286
287      for (unsigned int i = 0; i < N; i++)
288          accu += tables[_numObj][i][sigma(_numObj, _sol, (int)i)];
289
290      return accu / (double)N;
291  }
292
293  /*****
294  *
295  * Extract epistatic links of the fitness contribution i
296  *
297  * @param _numObj the objective function to consider
298  * @param _sol the solution to evaluate
299  * @param _i bit of the contribution
300  *
301  *****/
302  unsigned int sigma(unsigned _numObj, std::vector<bool> &_sol, int _i) {
303      unsigned int n = 1;
304      unsigned int accu = 0;
305
306      for (unsigned int j = 0; j < K + 1; j++) {
307          if (_sol[links[_numObj][_i][j]] == 1)
308              accu = accu | n;
309      }

```

```

310         n = n « 1;
311     }
312
313     return accu;
314 }
315 };
316
317 } // namespace pmnk
318 #endif // __rMNKEval

```

## 5.11 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/↵ Utils/solution.hpp File Reference

Implementation of a solution class.

```

#include <cassert>
#include <random>
#include "rMNKEval.hpp"

```

### Classes

- class [pmnk::Solution](#)  
*Standard solution class.*
- class [pmnk::GASolution](#)  
*Genetic algorithm solution wrapper (adds a fitness attribute to the [Solution](#) class)*

### Typedefs

- using **pmnk::DecisionVector** = std::vector< bool >
- using **pmnk::ObjectiveVector** = std::vector< double >

### Enumerations

- enum class **DominanceType** { **DOMINATES** , **EQUAL** , **DOMINATED** , **INCOMPARABLE** }

#### 5.11.1 Detailed Description

Implementation of a solution class.

##### Author

Pedro Rodrigues ( [pedror@student.dei.uc.pt](mailto:pedror@student.dei.uc.pt) )  
 Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

##### Version

0.1.0

##### Date

13-09-2021

##### Copyright

Copyright (c) 2021

## 5.12 solution.hpp

[Go to the documentation of this file.](#)

```

1
2 #ifndef SOLUTION_HPP
3 #define SOLUTION_HPP
4
5 #include <cassert>
6 #include <random>
7
8 #include "rMNKEval.hpp"
9
10 namespace pmnk {
11
12 using DecisionVector = std::vector<bool>;
13 using ObjectiveVector = std::vector<double>;
14
15 enum class DominanceType { DOMINATES, EQUAL, DOMINATED, INCOMPARABLE };
16
17 class Solution {
18 protected:
19     DecisionVector m_decision;
20     ObjectiveVector m_objective;
21
22 public:
23     explicit Solution() = default;
24
25     Solution(Solution const &other) = default;
26
27     Solution(Solution &&other) = default;
28
29     Solution &operator=(Solution const &other) = default;
30
31     Solution &operator=(Solution &&other) = default;
32
33     Solution(RMNKEval &rmnk, DecisionVector const &decision)
34         : m_decision(decision) {
35         eval(rmnk);
36     }
37
38     Solution(RMNKEval &rmnk, DecisionVector &&decision)
39         : m_decision(std::move(decision)) {
40         eval(rmnk);
41     }
42
43     DecisionVector const &decision_vector() const {
44         return m_decision;
45     }
46
47     ObjectiveVector const &objective_vector() const {
48         return m_objective;
49     }
50
51     std::size_t size() const noexcept {
52         return m_decision.size();
53     };
54
55     DominanceType dominance(Solution const &solution) const {
56         assert(m_decision.size() == solution.m_decision.size());
57
58         DominanceType res = DominanceType::EQUAL;
59         for (decltype(m_objective.size()) i = 0; i < m_objective.size(); ++i) {
60             if (m_objective[i] < solution.m_objective[i]) {
61                 if (res == DominanceType::DOMINATES) {
62                     return DominanceType::INCOMPARABLE;
63                 }
64                 res = DominanceType::DOMINATED;
65             } else if (m_objective[i] > solution.m_objective[i]) {
66                 if (res == DominanceType::DOMINATED) {
67                     return DominanceType::INCOMPARABLE;
68                 }
69                 res = DominanceType::DOMINATES;
70             }
71         }
72         return res;
73     }
74
75     friend std::ostream &operator<<(std::ostream &os, Solution const &solution) {
76         for (auto const &i : solution.objective_vector()) {
77             os << i << " ";
78         }
79         os << "\n";
80         return os;
81     }
82 }
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160

```

```

170 DecisionVector::reference operator[](std::size_t const i) {
171     return m_decision[i];
172 }
173
174 void eval(RMNKEval &rmnk) {
175     rmnk.eval(m_decision, m_objective);
176 }
177
178 template <typename RNG>
179 static Solution random_solution(RMNKEval &eval, RNG &generator) {
180     std::uniform_int_distribution<int> distrib(0, 1);
181
182     DecisionVector decision_vector(eval.getN(), 0);
183     for (std::size_t i = 0; i < decision_vector.size(); ++i)
184         decision_vector[i] = distrib(generator);
185
186     return Solution(eval, std::move(decision_vector));
187 }
188
189 template <typename RNG>
190 static Solution uniform_bit_flip_solution(RMNKEval &eval, RNG &generator,
191     Solution const &original) {
192     std::bernoulli_distribution distrib(1 / (double)original.decision_vector().size());
193
194     DecisionVector flipped = original.decision_vector();
195     for (decltype(flipped.size()) i = 0; i < flipped.size(); ++i) {
196         if (distrib(generator)) {
197             flipped[i] = !flipped[i];
198         }
199     }
200     return Solution(eval, std::move(flipped));
201 }
202
203 static std::vector<Solution> neighborhood_solutions(RMNKEval &eval, Solution const &original) {
204     std::vector<Solution> neighborhood;
205     neighborhood.reserve(original.decision_vector().size());
206
207     for (size_t i = 0; i < original.decision_vector().size(); ++i) {
208         auto decision_vector = original.decision_vector();
209         decision_vector[i] = !decision_vector[i];
210
211         neighborhood.emplace_back(eval, std::move(decision_vector));
212     }
213
214     for (size_t i = 0; i < original.decision_vector().size(); ++i) {
215         for (size_t j = i + 1; j < original.decision_vector().size(); ++j) {
216             if (original.decision_vector()[i] == original.decision_vector()[j]) {
217                 continue;
218             }
219
220             auto decision_vector = original.decision_vector();
221             swap(decision_vector[i], decision_vector[j]);
222             neighborhood.emplace_back(eval, std::move(decision_vector));
223         }
224     }
225     return neighborhood;
226 }
227
228 class GASolution : public Solution {
229     double m_fitness;
230
231 public:
232     explicit GASolution() = default;
233
234     GASolution(Solution &&sol, double fitness)
235         : Solution(std::forward<Solution>(sol))
236         , m_fitness(fitness) {}
237
238     GASolution(Solution &&sol)
239         : GASolution(std::forward<Solution>(sol), 0) {}
240
241     [[nodiscard]] constexpr double const &fitness() const {
242         return m_fitness;
243     }
244
245     void set_objv(ObjectiveVector &&objv) {
246         m_objective = std::move(objv);
247     }
248
249     constexpr void set_fitness(double const fitness) {
250         m_fitness = fitness;
251     }
252 };
253
254 } // namespace pmnk
255 #endif // SOLUTION_HPP

```

## 5.13 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/Utils.hpp File Reference ↩

Project Utility functions.

```
#include "solution.hpp"
```

### Functions

- `template<typename Vec , typename S >`  
`bool pmnk::add_non_dominated (Vec &solutions, S &&solution)`  
*Utility function responsible for maintaining a container of non-dominated solutions.*

#### 5.13.1 Detailed Description

Project Utility functions.

##### Author

Pedro Rodrigues ( [pedror@student.dei.uc.pt](mailto:pedror@student.dei.uc.pt) )  
 Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

##### Version

0.1.0

##### Date

13-09-2021

##### Copyright

Copyright (c) 2021

#### 5.13.2 Function Documentation

##### 5.13.2.1 add\_non\_dominated()

```
template<typename Vec , typename S >
bool pmnk::add_non_dominated (
    Vec & solutions,
    S && solution )
```

Utility function responsible for maintaining a container of non-dominated solutions.

### Template Parameters

|            |  |
|------------|--|
| <i>Vec</i> | The type for the container holding the solutions.              |
| <i>S</i>   | The type for a solution to be added to the solution container. |

### Parameters

|                  |  |
|------------------|--|
| <i>solutions</i> | The container for the non-dominated solutions. |
| <i>solution</i>  | The solution to be added to the container.     |

### Returns

true If the solution was successfully added to the container

false If the solution to be added is dominated by other solution already present and failed to be inserted.

## 5.14 utils.hpp

[Go to the documentation of this file.](#)

```

1
13 #ifndef UTILS_HPP
14 #define UTILS_HPP
15
16 #include "solution.hpp"
17
18 namespace pmnk {
19
32 template <typename Vec, typename S>
33 bool add_non_dominated(Vec &solutions, S &&solution) {
34     for (std::size_t i = 0; i < solutions.size(); i) {
35         auto d = solution.dominance(solutions[i]);
36         if (d == DominanceType::EQUAL) {
37             if (solution.decision_vector() == solutions[i].decision_vector()) {
38                 return false;
39             } else {
40                 for (++i; i < solutions.size(); ++i) {
41                     if (solution.decision_vector() == solutions[i].decision_vector()) {
42                         return false;
43                     }
44                 }
45                 break;
46             }
47         } else if (d == DominanceType::DOMINATES) {
48             solutions[i] = std::move(solutions.back());
49             solutions.pop_back();
50         } else if (d == DominanceType::DOMINATED) {
51             return false;
52         } else {
53             ++i;
54         }
55     }
56     solutions.push_back(std::forward<S>(solution));
57     return true;
58 }
59 } // namespace pmnk
60 #endif // UTILS_HPP

```

## 5.15 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/↵ Utils/wfg.hpp File Reference

Implementation of the wfg algorithm to calculate hypervolumes.



```
#include <algorithm>
#include <array>
#include <limits>
#include <type_traits>
#include <vector>
```

## Classes

- class [pmnk::hvobj< T >](#)  
*Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm)*

## Functions

- template<typename T >  
auto [pmnk::weakly\\_dominates](#) (T const &lhs, T const &rhs)  
*Check pareto weakly dominance between two points.*
- template<typename T >  
void [pmnk::insert\\_non\\_dominated](#) (T &&sol, std::vector< T > &set)  
*Insert a point (solution) into a non dominated (solution) set of points.*
- template<typename Iter , typename T >  
auto [pmnk::limit\\_set](#) (Iter begin, Iter end, T const &sol)  
*Limit the set of points by replacing them with the points whose value in each objective is limited to be no better than the contributing point.*
- template<typename T , typename R >  
auto [pmnk::point\\_hv](#) (T const &p, R const &r)  
*Compute hypervolume of a point in relation to a reference.*
- template<typename S , typename T >  
std::common\_type\_t< typename S::value\_type::value\_type, typename T::value\_type > [pmnk::set\\_hv\\_wfg](#) (S const &s, T const &ref)  
*Compute a set hypervolume value given a reference point using the wfg algorithm. (Worker function)*
- template<typename S , typename T >  
auto [pmnk::set\\_hv](#) (S const &s, T const &ref)  
*Compute a set hypervolume value given a reference point using the wfg algorithm. (Wrapper function)*
- template<typename T , typename S , typename R >  
auto [pmnk::point\\_hvc](#) (T const &p, S const &s, R const &ref)  
*Calculate a point hypervolume contribution to a set.*

### 5.15.1 Detailed Description

Implementation of the wfg algorithm to calculate hypervolumes.

#### Author

Alexandre Jesus ( [ajesus@dei.uc.pt](mailto:ajesus@dei.uc.pt) )

#### Date

13-09-2021

#### Copyright

Copyright (c) 2021

## 5.15.2 Function Documentation

### 5.15.2.1 insert\_non\_dominated()

```
template<typename T >
void pmnk::insert_non_dominated (
    T && sol,
    std::vector< T > & set )
```

Insert a point (solution) into a non dominated (solution) set of points.

#### Template Parameters

|          |   |
|----------|---|
| <i>T</i> | The type for the point being supplied as a parameter. |
|----------|---|

#### Parameters

|            |  |
|------------|--|
| <i>sol</i> | The point (solution) to be added to the set            |
| <i>set</i> | The (solution) set of non dominated points (solutions) |

### 5.15.2.2 limit\_set()

```
template<typename Iter , typename T >
auto pmnk::limit_set (
    Iter begin,
    Iter end,
    T const & sol )
```

Limit the set of points by replacing them with the points whose value in each objective is limited to be no better than the contributing point.

#### Template Parameters

|             |   |
|-------------|---|
| <i>Iter</i> | The type for and iterator of a set of points                      |
| <i>T</i>    | The type for the contributing point being supplied as a parameter |

#### Parameters

|              |  |
|--------------|--|
| <i>begin</i> | An iterator for the beginning of the set |
| <i>end</i>   | An iterator for the end of the set       |
| <i>sol</i>   | The contributing solution                |

**Returns**

auto The limited set

**5.15.2.3 point\_hv()**

```
template<typename T , typename R >
auto pmnk::point_hv (
    T const & p,
    R const & r )
```

Compute hypervolume of a point in relation to a reference.

**Template Parameters**

|          |  |
|----------|--|
| <i>T</i> | The type for the point being supplied as a parameter     |
| <i>R</i> | The type for the reference being supplied as a parameter |

**Parameters**

|          |   |
|----------|---|
| <i>p</i> | The point whose hypervolume value is to be computed |
| <i>r</i> | The reference point.                                |

**Returns**

auto The hypervolume value.

**5.15.2.4 point\_hvc()**

```
template<typename T , typename S , typename R >
auto pmnk::point_hvc (
    T const & p,
    S const & s,
    R const & ref )
```

Calculate a point hypervolume contribution to a set.

**Template Parameters**

|          |   |
|----------|---|
| <i>S</i> | The type for the set supplied as a parameter.             |
| <i>T</i> | The type for the point being supplied as a parameter.     |
| <i>R</i> | The type for the reference point supplied as a parameter. |

**Parameters**

|            |   |
|------------|---|
| <i>p</i>   | The point whose contribution is going to be calculated. |
| <i>s</i>   | The solution set.                                       |
| <i>ref</i> | The reference point.                                    |

**Returns**

auto The point hypervolume contribution.

**5.15.2.5 set\_hv()**

```
template<typename S , typename T >
auto pmnk::set_hv (
    S const & s,
    T const & ref )
```

Compute a set hypervolume value given a reference point using the wfg algorithm. (Wrapper function)

**Template Parameters**

|          |  |
|----------|--|
| <i>S</i> | The type for the set supplied as a parameter         |
| <i>T</i> | The type for the point being supplied as a parameter |

**Parameters**

|            |                      |
|------------|----------------------|
| <i>s</i>   | The solution set     |
| <i>ref</i> | The reference point. |

**Returns**

std::common\_type\_t<typename S::value\_type::value\_type, typename T::value\_type> The resulting set hypervolume value.

**5.15.2.6 set\_hv\_wfg()**

```
template<typename S , typename T >
std::common_type_t< typename S::value_type::value_type, typename T::value_type > pmnk::set_hv_wfg (
    S const & s,
    T const & ref )
```

Compute a set hypervolume value given a reference point using the wfg algorithm. (Worker function)

## Template Parameters

|          |  |
|----------|--|
| <i>S</i> | The type for the set supplied as a parameter         |
| <i>T</i> | The type for the point being supplied as a parameter |

## Parameters

|            |                      |
|------------|----------------------|
| <i>s</i>   | The solution set     |
| <i>ref</i> | The reference point. |

## Returns

`std::common_type_t<typename S::value_type::value_type, typename T::value_type>` The resulting set hypervolume value.

## 5.15.2.7 weakly\_dominates()

```
template<typename T >
auto pmnk::weakly_dominates (
    T const & lhs,
    T const & rhs )
```

Check pareto weakly dominance between two points.

## Template Parameters

|          |   |
|----------|---|
| <i>T</i> | The type for the point being supplied as a parameter. |
|----------|---|

## Parameters

|            |   |
|------------|---|
| <i>lhs</i> | The first point (left hand side point)  |
| <i>rhs</i> | The second point (left hand side point) |

## Returns

`auto` A boolean with the value true if the left hand side point (lhs) weakly dominated the right hand side point (rhs)

## 5.16 wfg.hpp

[Go to the documentation of this file.](#)

```
1
9 #ifndef WFG_H
10 #define WFG_H
11
12 #include <algorithm>
13 #include <array>
```

```

14 #include <limits>
15 #include <type_traits>
16 #include <vector>
17
18 // This code assumes maximizing objective functions
19
20 namespace pmnk {
21
22 template <typename T>
23 auto weakly_dominates(T const& lhs, T const& rhs) {
24     for (decltype(lhs.size()) i = 0; i < lhs.size(); ++i) {
25         if (lhs[i] < rhs[i]) {
26             return false;
27         }
28     }
29     return true;
30 }
31
32 template <typename T>
33 void insert_non_dominated(T&& sol, std::vector<T>& set) {
34     for (auto it = set.begin(); it != set.end(); ++it) {
35         if (weakly_dominates(*it, sol)) {
36             return;
37         } else if (weakly_dominates(sol, *it)) {
38             *it = std::move(set.back());
39             set.pop_back();
40             set.erase(
41                 std::remove_if(it, set.end(), [&sol](auto const& s) { return weakly_dominates(sol, s); }));
42             set.end();
43             break;
44         }
45     }
46     set.push_back(std::move(sol));
47 }
48
49 template <typename Iter, typename T>
50 auto limit_set(Iter begin, Iter end, T const& sol) {
51     std::vector<T> res;
52     res.reserve(static_cast<typename std::vector<T>::size_type>(std::distance(begin, end)));
53     for (; begin != end; ++begin) {
54         auto aux = *begin;
55         for (size_t i = 0; i < sol.size(); ++i) {
56             aux[i] = std::min(aux[i], sol[i]);
57         }
58         insert_non_dominated(std::move(aux), res);
59     }
60     return res;
61 }
62
63 template <typename T, typename R>
64 auto point_hv(T const& p, R const& r) {
65     auto res = p[0] - r[0];
66     for (size_t i = 1; i < p.size(); ++i) {
67         res *= p[i] - r[i];
68     }
69     return res;
70 }
71
72 template <typename S, typename T>
73 std::common_type_t<typename S::value_type::value_type, typename T::value_type> set_hv_wfg(
74     S const& s, T const& ref) {
75     using result_t = std::common_type_t<typename S::value_type::value_type, typename T::value_type>;
76     auto res = result_t(0);
77     for (auto it = s.begin(); it != s.end(); ++it) {
78         res += point_hv(*it, ref) - set_hv_wfg(limit_set(std::next(it), s.end(), *it), ref);
79     }
80     return res;
81 }
82
83 template <typename S, typename T>
84 auto set_hv(S const& s, T const& ref) {
85     std::vector<T> v;
86     v.reserve(s.size());
87     for (auto const& sol : s) {
88         v.push_back(sol.objective_vector());
89     }
90     sort(v.begin(), v.end(), [](auto const& a, auto const& b) { return a[0] < b[0]; });
91     return set_hv_wfg(v, ref);
92 }
93
94 template <typename T, typename S, typename R>
95 auto point_hvc(T const& p, S const& s, R const& ref) {
96     std::vector<T> v;
97     v.reserve(s.size());
98     for (auto const& sol : s) {
99         v.push_back(sol.objective_vector());
100     }

```

```

172     sort(v.begin(), v.end(), [](auto const& a, auto const& b) { return a[0] < b[0]; });
173     return point_hv(p, ref) - set_hv_wfg(limit_set(v.begin(), v.end(), p), ref);
174 }
175
176 template <typename T>
177 class [[nodiscard]] hvobj {
178 public:
179     using hv_type = T;
180     using ovec_type = std::vector<hv_type>;
181     using set_type = std::vector<ovec_type>;
182
183     constexpr explicit hvobj(ovec_type const& r)
184         : m_hv(0)
185         , m_set()
186         , m_ref(r) {}
187
188     constexpr hvobj(hvobj const& other) = default;
189     constexpr hvobj(hvobj&& other) noexcept = default;
190
191     [[nodiscard]] constexpr auto value() const {
192         return m_hv;
193     }
194
195     template <typename V>
196     [[nodiscard]] constexpr auto contribution(V const& v) const {
197         return m_point_hv(v, m_ref) - m_set_hv(m_limit_set(m_set, v), m_ref);
198     }
199
200     template <typename V>
201     constexpr auto insert(V&& v) {
202         auto hvc = contribution(v);
203         if (hvc != 0) {
204             m_insert_non_dominated(std::forward<V>(v), m_set);
205             m_hv += hvc;
206         }
207         return hvc;
208     }
209
210     template <typename V>
211     constexpr auto remove(V const& v) {
212         auto it = std::find(m_set.begin(), m_set.end(), v);
213         if (it == m_set.end())
214             return -1.0;
215         m_set.erase(it);
216         auto hvc = contribution(v);
217         m_hv -= hvc;
218         return hvc;
219     }
220
221 private:
222     template <typename V>
223     [[nodiscard]] constexpr auto m_weakly_dominates(V const& a, V const& b) const {
224         for (size_t i = 1; i < a.size(); ++i) {
225             if (a[i] < b[i]) {
226                 return false;
227             }
228         }
229         return true;
230     }
231
232     template <typename V, typename C>
233     void m_insert_non_dominated(V&& v, C& set) const {
234         auto it = set.begin();
235         for (; it != set.end() && (*it)[0] > v[0]; ++it) {
236             if (m_weakly_dominates(*it, v)) {
237                 return;
238             }
239         }
240         for (; it != set.end() && (*it)[0] == v[0]; ++it) {
241             if (m_weakly_dominates(*it, v)) {
242                 return;
243             } else if (m_weakly_dominates(v, *it)) {
244                 *it = std::forward<V>(v);
245                 set.erase(std::remove_if(std::next(it), set.end(),
246                                         [this, it](auto const& a) { return m_weakly_dominates(*it, a); }),
247                         set.end());
248                 return;
249             }
250         }
251         if (it == set.end()) {
252             set.push_back(std::forward<V>(v));
253         } else {
254             auto aux = std::forward<V>(v);
255             std::swap(aux, *it);
256         }
257     }
258
259     template <typename V>
260     void m_limit_set(set_type& s, V const& v) const {
261         auto it = s.begin();
262         for (; it != s.end(); ++it) {
263             if (*it > v) {
264                 it = s.erase(it);
265             }
266         }
267     }
268
269     template <typename V>
270     void m_set_hv(set_type& s) const {
271         auto hvc = 0.0;
272         for (const auto& ovec : s) {
273             hvc += ovec[0];
274         }
275     }
276
277     template <typename V>
278     set_type m_limit_set(set_type& s, V const& v) const {
279         m_limit_set(s, v);
280         return s;
281     }
282
283     template <typename V>
284     void m_set_hv(set_type& s) const {
285         m_set_hv(s);
286     }
287
288     template <typename V>
289     void m_insert_non_dominated(V&& v, set_type& s) const {
290         m_insert_non_dominated(v, s);
291     }
292
293     template <typename V>
294     void m_remove(V const& v, set_type& s) const {
295         m_remove(v, s);
296     }
297
298     template <typename V>
299     void m_hv += hvc;
300
301     template <typename V>
302     void m_hv -= hvc;
303
304     template <typename V>
305     void m_hv = hvc;
306
307     template <typename V>
308     void m_hv *= hvc;
309
310     template <typename V>
311     void m_hv /= hvc;
312
313     template <typename V>
314     void m_hv %= hvc;
315
316     template <typename V>
317     void m_hv *= hvc;
318
319     template <typename V>
320     void m_hv /= hvc;
321
322     template <typename V>
323     void m_hv %= hvc;
324
325     template <typename V>
326     void m_hv *= hvc;
327
328     template <typename V>
329     void m_hv /= hvc;
330
331     template <typename V>
332     void m_hv %= hvc;
333
334     template <typename V>
335     void m_hv *= hvc;
336
337     template <typename V>
338     void m_hv /= hvc;
339
340     template <typename V>
341     void m_hv %= hvc;
342
343     template <typename V>
344     void m_hv *= hvc;
345
346     template <typename V>
347     void m_hv /= hvc;
348
349     template <typename V>
350     void m_hv %= hvc;
351
352     template <typename V>
353     void m_hv *= hvc;
354
355     template <typename V>
356     void m_hv /= hvc;
357
358     template <typename V>
359     void m_hv %= hvc;
360
361     template <typename V>
362     void m_hv *= hvc;
363
364     template <typename V>
365     void m_hv /= hvc;
366
367     template <typename V>
368     void m_hv %= hvc;
369
370     template <typename V>
371     void m_hv *= hvc;
372
373     template <typename V>
374     void m_hv /= hvc;
375
376     template <typename V>
377     void m_hv %= hvc;
378
379     template <typename V>
380     void m_hv *= hvc;
381
382     template <typename V>
383     void m_hv /= hvc;
384
385     template <typename V>
386     void m_hv %= hvc;
387
388     template <typename V>
389     void m_hv *= hvc;
390
391     template <typename V>
392     void m_hv /= hvc;
393
394     template <typename V>
395     void m_hv %= hvc;
396
397     template <typename V>
398     void m_hv *= hvc;
399
400     template <typename V>
401     void m_hv /= hvc;
402
403     template <typename V>
404     void m_hv %= hvc;
405
406     template <typename V>
407     void m_hv *= hvc;
408
409     template <typename V>
410     void m_hv /= hvc;
411
412     template <typename V>
413     void m_hv %= hvc;
414
415     template <typename V>
416     void m_hv *= hvc;
417
418     template <typename V>
419     void m_hv /= hvc;
420
421     template <typename V>
422     void m_hv %= hvc;
423
424     template <typename V>
425     void m_hv *= hvc;
426
427     template <typename V>
428     void m_hv /= hvc;
429
430     template <typename V>
431     void m_hv %= hvc;
432
433     template <typename V>
434     void m_hv *= hvc;
435
436     template <typename V>
437     void m_hv /= hvc;
438
439     template <typename V>
440     void m_hv %= hvc;
441
442     template <typename V>
443     void m_hv *= hvc;
444
445     template <typename V>
446     void m_hv /= hvc;
447
448     template <typename V>
449     void m_hv %= hvc;
450
451     template <typename V>
452     void m_hv *= hvc;
453
454     template <typename V>
455     void m_hv /= hvc;
456
457     template <typename V>
458     void m_hv %= hvc;
459
460     template <typename V>
461     void m_hv *= hvc;
462
463     template <typename V>
464     void m_hv /= hvc;
465
466     template <typename V>
467     void m_hv %= hvc;
468
469     template <typename V>
470     void m_hv *= hvc;
471
472     template <typename V>
473     void m_hv /= hvc;
474
475     template <typename V>
476     void m_hv %= hvc;
477
478     template <typename V>
479     void m_hv *= hvc;
480
481     template <typename V>
482     void m_hv /= hvc;
483
484     template <typename V>
485     void m_hv %= hvc;
486
487     template <typename V>
488     void m_hv *= hvc;
489
490     template <typename V>
491     void m_hv /= hvc;
492
493     template <typename V>
494     void m_hv %= hvc;
495
496     template <typename V>
497     void m_hv *= hvc;
498
499     template <typename V>
500     void m_hv /= hvc;
501
502     template <typename V>
503     void m_hv %= hvc;
504
505     template <typename V>
506     void m_hv *= hvc;
507
508     template <typename V>
509     void m_hv /= hvc;
510
511     template <typename V>
512     void m_hv %= hvc;
513
514     template <typename V>
515     void m_hv *= hvc;
516
517     template <typename V>
518     void m_hv /= hvc;
519
520     template <typename V>
521     void m_hv %= hvc;
522
523     template <typename V>
524     void m_hv *= hvc;
525
526     template <typename V>
527     void m_hv /= hvc;
528
529     template <typename V>
530     void m_hv %= hvc;
531
532     template <typename V>
533     void m_hv *= hvc;
534
535     template <typename V>
536     void m_hv /= hvc;
537
538     template <typename V>
539     void m_hv %= hvc;
540
541     template <typename V>
542     void m_hv *= hvc;
543
544     template <typename V>
545     void m_hv /= hvc;
546
547     template <typename V>
548     void m_hv %= hvc;
549
550     template <typename V>
551     void m_hv *= hvc;
552
553     template <typename V>
554     void m_hv /= hvc;
555
556     template <typename V>
557     void m_hv %= hvc;
558
559     template <typename V>
560     void m_hv *= hvc;
561
562     template <typename V>
563     void m_hv /= hvc;
564
565     template <typename V>
566     void m_hv %= hvc;
567
568     template <typename V>
569     void m_hv *= hvc;
570
571     template <typename V>
572     void m_hv /= hvc;
573
574     template <typename V>
575     void m_hv %= hvc;
576
577     template <typename V>
578     void m_hv *= hvc;
579
580     template <typename V>
581     void m_hv /= hvc;
582
583     template <typename V>
584     void m_hv %= hvc;
585
586     template <typename V>
587     void m_hv *= hvc;
588
589     template <typename V>
590     void m_hv /= hvc;
591
592     template <typename V>
593     void m_hv %= hvc;
594
595     template <typename V>
596     void m_hv *= hvc;
597
598     template <typename V>
599     void m_hv /= hvc;
600
601     template <typename V>
602     void m_hv %= hvc;
603
604     template <typename V>
605     void m_hv *= hvc;
606
607     template <typename V>
608     void m_hv /= hvc;
609
610     template <typename V>
611     void m_hv %= hvc;
612
613     template <typename V>
614     void m_hv *= hvc;
615
616     template <typename V>
617     void m_hv /= hvc;
618
619     template <typename V>
620     void m_hv %= hvc;
621
622     template <typename V>
623     void m_hv *= hvc;
624
625     template <typename V>
626     void m_hv /= hvc;
627
628     template <typename V>
629     void m_hv %= hvc;
630
631     template <typename V>
632     void m_hv *= hvc;
633
634     template <typename V>
635     void m_hv /= hvc;
636
637     template <typename V>
638     void m_hv %= hvc;
639
640     template <typename V>
641     void m_hv *= hvc;
642
643     template <typename V>
644     void m_hv /= hvc;
645
646     template <typename V>
647     void m_hv %= hvc;
648
649     template <typename V>
650     void m_hv *= hvc;
651
652     template <typename V>
653     void m_hv /= hvc;
654
655     template <typename V>
656     void m_hv %= hvc;
657
658     template <typename V>
659     void m_hv *= hvc;
660
661     template <typename V>
662     void m_hv /= hvc;
663
664     template <typename V>
665     void m_hv %= hvc;
666
667     template <typename V>
668     void m_hv *= hvc;
669
670     template <typename V>
671     void m_hv /= hvc;
672
673     template <typename V>
674     void m_hv %= hvc;
675
676     template <typename V>
677     void m_hv *= hvc;
678
679     template <typename V>
680     void m_hv /= hvc;
681
682     template <typename V>
683     void m_hv %= hvc;
684
685     template <typename V>
686     void m_hv *= hvc;
687
688     template <typename V>
689     void m_hv /= hvc;
690
691     template <typename V>
692     void m_hv %= hvc;
693
694     template <typename V>
695     void m_hv *= hvc;
696
697     template <typename V>
698     void m_hv /= hvc;
699
700     template <typename V>
701     void m_hv %= hvc;
702
703     template <typename V>
704     void m_hv *= hvc;
705
706     template <typename V>
707     void m_hv /= hvc;
708
709     template <typename V>
710     void m_hv %= hvc;
711
712     template <typename V>
713     void m_hv *= hvc;
714
715     template <typename V>
716     void m_hv /= hvc;
717
718     template <typename V>
719     void m_hv %= hvc;
720
721     template <typename V>
722     void m_hv *= hvc;
723
724     template <typename V>
725     void m_hv /= hvc;
726
727     template <typename V>
728     void m_hv %= hvc;
729
730     template <typename V>
731     void m_hv *= hvc;
732
733     template <typename V>
734     void m_hv /= hvc;
735
736     template <typename V>
737     void m_hv %= hvc;
738
739     template <typename V>
740     void m_hv *= hvc;
741
742     template <typename V>
743     void m_hv /= hvc;
744
745     template <typename V>
746     void m_hv %= hvc;
747
748     template <typename V>
749     void m_hv *= hvc;
750
751     template <typename V>
752     void m_hv /= hvc;
753
754     template <typename V>
755     void m_hv %= hvc;
756
757     template <typename V>
758     void m_hv *= hvc;
759
760     template <typename V>
761     void m_hv /= hvc;
762
763     template <typename V>
764     void m_hv %= hvc;
765
766     template <typename V>
767     void m_hv *= hvc;
768
769     template <typename V>
770     void m_hv /= hvc;
771
772     template <typename V>
773     void m_hv %= hvc;
774
775     template <typename V>
776     void m_hv *= hvc;
777
778     template <typename V>
779     void m_hv /= hvc;
780
781     template <typename V>
782     void m_hv %= hvc;
783
784     template <typename V>
785     void m_hv *= hvc;
786
787     template <typename V>
788     void m_hv /= hvc;
789
790     template <typename V>
791     void m_hv %= hvc;
792
793     template <typename V>
794     void m_hv *= hvc;
795
796     template <typename V>
797     void m_hv /= hvc;
798
799     template <typename V>
800     void m_hv %= hvc;
801
802     template <typename V>
803     void m_hv *= hvc;
804
805     template <typename V>
806     void m_hv /= hvc;
807
808     template <typename V>
809     void m_hv %= hvc;
810
811     template <typename V>
812     void m_hv *= hvc;
813
814     template <typename V>
815     void m_hv /= hvc;
816
817     template <typename V>
818     void m_hv %= hvc;
819
820     template <typename V>
821     void m_hv *= hvc;
822
823     template <typename V>
824     void m_hv /= hvc;
825
826     template <typename V>
827     void m_hv %= hvc;
828
829     template <typename V>
830     void m_hv *= hvc;
831
832     template <typename V>
833     void m_hv /= hvc;
834
835     template <typename V>
836     void m_hv %= hvc;
837
838     template <typename V>
839     void m_hv *= hvc;
840
841     template <typename V>
842     void m_hv /= hvc;
843
844     template <typename V>
845     void m_hv %= hvc;
846
847     template <typename V>
848     void m_hv *= hvc;
849
850     template <typename V>
851     void m_hv /= hvc;
852
853     template <typename V>
854     void m_hv %= hvc;
855
856     template <typename V>
857     void m_hv *= hvc;
858
859     template <typename V>
860     void m_hv /= hvc;
861
862     template <typename V>
863     void m_hv %= hvc;
864
865     template <typename V>
866     void m_hv *= hvc;
867
868     template <typename V>
869     void m_hv /= hvc;
870
871     template <typename V>
872     void m_hv %= hvc;
873
874     template <typename V>
875     void m_hv *= hvc;
876
877     template <typename V>
878     void m_hv /= hvc;
879
880     template <typename V>
881     void m_hv %= hvc;
882
883     template <typename V>
884     void m_hv *= hvc;
885
886     template <typename V>
887     void m_hv /= hvc;
888
889     template <typename V>
890     void m_hv %= hvc;
891
892     template <typename V>
893     void m_hv *= hvc;
894
895     template <typename V>
896     void m_hv /= hvc;
897
898     template <typename V>
899     void m_hv %= hvc;
900
901     template <typename V>
902     void m_hv *= hvc;
903
904     template <typename V>
905     void m_hv /= hvc;
906
907     template <typename V>
908     void m_hv %= hvc;
909
910     template <typename V>
911     void m_hv *= hvc;
912
913     template <typename V>
914     void m_hv /= hvc;
915
916     template <typename V>
917     void m_hv %= hvc;
918
919     template <typename V>
920     void m_hv *= hvc;
921
922     template <typename V>
923     void m_hv /= hvc;
924
925     template <typename V>
926     void m_hv %= hvc;
927
928     template <typename V>
929     void m_hv *= hvc;
930
931     template <typename V>
932     void m_hv /= hvc;
933
934     template <typename V>
935     void m_hv %= hvc;
936
937     template <typename V>
938     void m_hv *= hvc;
939
940     template <typename V>
941     void m_hv /= hvc;
942
943     template <typename V>
944     void m_hv %= hvc;
945
946     template <typename V>
947     void m_hv *= hvc;
948
949     template <typename V>
950     void m_hv /= hvc;
951
952     template <typename V>
953     void m_hv %= hvc;
954
955     template <typename V>
956     void m_hv *= hvc;
957
958     template <typename V>
959     void m_hv /= hvc;
960
961     template <typename V>
962     void m_hv %= hvc;
963
964     template <typename V>
965     void m_hv *= hvc;
966
967     template <typename V>
968     void m_hv /= hvc;
969
970     template <typename V>
971     void m_hv %= hvc;
972
973     template <typename V>
974     void m_hv *= hvc;
975
976     template <typename V>
977     void m_hv /= hvc;
978
979     template <typename V>
980     void m_hv %= hvc;
981
982     template <typename V>
983     void m_hv *= hvc;
984
985     template <typename V>
986     void m_hv /= hvc;
987
988     template <typename V>
989     void m_hv %= hvc;
990
991     template <typename V>
992     void m_hv *= hvc;
993
994     template <typename V>
995     void m_hv /= hvc;
996
997     template <typename V>
998     void m_hv %= hvc;
999
1000    template <typename V>
1001    void m_hv *= hvc;
1002
1003    template <typename V>
1004    void m_hv /= hvc;
1005
1006    template <typename V>
1007    void m_hv %= hvc;
1008
1009    template <typename V>
1010    void m_hv *= hvc;
1011
1012    template <typename V>
1013    void m_hv /= hvc;
1014
1015    template <typename V>
1016    void m_hv %= hvc;
1017
1018    template <typename V>
1019    void m_hv *= hvc;
1020
1021    template <typename V>
1022    void m_hv /= hvc;
1023
1024    template <typename V>
1025    void m_hv %= hvc;
1026
1027    template <typename V>
1028    void m_hv *= hvc;
1029
1030    template <typename V>
1031    void m_hv /= hvc;
1032
1033    template <typename V>
1034    void m_hv %= hvc;
1035
1036    template <typename V>
1037    void m_hv *= hvc;
1038
1039    template <typename V>
1040    void m_hv /= hvc;
1041
1042    template <typename V>
1043    void m_hv %= hvc;
1044
1045    template <typename V>
1046    void m_hv *= hvc;
1047
1048    template <typename V>
1049    void m_hv /= hvc;
1050
1051    template <typename V>
1052    void m_hv %= hvc;
1053
1054    template <typename V>
1055    void m_hv *= hvc;
1056
1057    template <typename V>
1058    void m_hv /= hvc;
1059
1060    template <typename V>
1061    void m_hv %= hvc;
1062
1063    template <typename V>
1064    void m_hv *= hvc;
1065
1066    template <typename V>
1067    void m_hv /= hvc;
1068
1069    template <typename V>
1070    void m_hv %= hvc;
1071
1072    template <typename V>
1073    void m_hv *= hvc;
1074
1075    template <typename V>
1076    void m_hv /= hvc;
1077
1078    template <typename V>
1079    void m_hv %= hvc;
1080
1081    template <typename V>
1082    void m_hv *= hvc;
1083
1084    template <typename V>
1085    void m_hv /= hvc;
1086
1087    template <typename V>
1088    void m_hv %= hvc;
1089
1090    template <typename V>
1091    void m_hv *= hvc;
1092
1093    template <typename V>
1094    void m_hv /= hvc;
1095
1096    template <typename V>
1097    void m_hv %= hvc;
1098
1099    template <typename V>
1100    void m_hv *= hvc;
1101
1102    template <typename V>
1103    void m_hv /= hvc;
1104
1105    template <typename V>
1106    void m_hv %= hvc;
1107
1108    template <typename V>
1109    void m_hv *= hvc;
1110
1111    template <typename V>
1112    void m_hv /= hvc;
1113
1114    template <typename V>
1115    void m_hv %= hvc;
1116
1117    template <typename V>
1118    void m_hv *= hvc;
1119
1120    template <typename V>
1121    void m_hv /= hvc;
1122
1123    template <typename V>
1124    void m_hv %= hvc;
1125
1126    template <typename V>
1127    void m_hv *= hvc;
1128
1129    template <typename V>
1130    void m_hv /= hvc;
1131
1132    template <typename V>
1133    void m_hv %= hvc;
1134
1135    template <typename V>
1136    void m_hv *= hvc;
1137
1138    template <typename V>
1139    void m_hv /= hvc;
1140
1141    template <typename V>
1142    void m_hv %= hvc;
1143
1144    template <typename V>
1145    void m_hv *= hvc;
1146
1147    template <typename V>
1148    void m_hv /= hvc;
1149
1150    template <typename V>
1151    void m_hv %= hvc;
1152
1153    template <typename V>
1154    void m_hv *= hvc;
1155
1156    template <typename V>
1157    void m_hv /= hvc;
1158
1159    template <typename V>
1160    void m_hv %= hvc;
1161
1162    template <typename V>
1163    void m_hv *= hvc;
1164
1165    template <typename V>
1166    void m_hv /= hvc;
1167
1168    template <typename V>
1169    void m_hv %= hvc;
1170
1171    template <typename V>
1172    void m_hv *= hvc;
1173
1174    template <typename V>
1175    void m_hv /= hvc;
1176
1177    template <typename V>
1178    void m_hv %= hvc;
1179
1180    template <typename V>
1181    void m_hv *= hvc;
1182
1183    template <typename V>
1184    void m_hv /= hvc;
1185
1186    template <typename V>
1187    void m_hv %= hvc;
1188
1189    template <typename V>
1190    void m_hv *= hvc;
1191
1192    template <typename V>
1193    void m_hv /= hvc;
1194
1195    template <typename V>
1196    void m_hv %= hvc;
1197
1198    template <typename V>
1199    void m_hv *= hvc;
1200
1201    template <typename V>
1202    void m_hv /= hvc;
1203
1204    template <typename V>
1205    void m_hv %= hvc;
1206
1207    template <typename V>
1208    void m_hv *= hvc;
1209
1210    template <typename V>
1211    void m_hv /= hvc;
1212
1213    template <typename V>
1214    void m_hv %= hvc;
1215
1216    template <typename V>
1217    void m_hv *= hvc;
1218
1219    template <typename V>
1220    void m_hv /= hvc;
1221
1222    template <typename V>
1223    void m_hv %= hvc;
1224
1225    template <typename V>
1226    void m_hv *= hvc;
1227
1228    template <typename V>
1229    void m_hv /= hvc;
1230
1231    template <typename V>
1232    void m_hv %= hvc;
1233
1234    template <typename V>
1235    void m_hv *= hvc;
1236
1237    template <typename V>
1238    void m_hv /= hvc;
1239
1240    template <typename V>
1241    void m_hv %= hvc;
1242
1243    template <typename V>
1244    void m_hv *= hvc;
1245
1246    template <typename V>
1247    void m_hv /= hvc;
1248
1249    template <typename V>
1250    void m_hv %= hvc;
1251
1252    template <typename V>
1253    void m_hv *= hvc;
1254
1255    template <typename V>
1256    void m_hv /= hvc;
1257
1258    template <typename V>
1259    void m_hv %= hvc;
1260
1261    template <typename V>
1262    void m_hv *= hvc;
1263
1264    template <typename V>
1265    void m_hv /= hvc;
1266
1267    template <typename V>
1268    void m_hv %= hvc;
1269
1270    template <typename V>
1271    void m_hv *= hvc;
1272
1273    template <typename V>
1274    void m_hv /= hvc;
1275
1276    template <typename V>
1277    void m_hv %= hvc;
1278
1279    template <typename V>
1280    void m_hv *= hvc;
1281
1282    template <typename V>
1283    void m_hv /= hvc;
1284
1285    template <typename V>
1286    void m_hv %= hvc;
1287
1288    template <typename V>
1289    void m_hv *= hvc;
1290
1291    template <typename V>
1292    void m_hv /= hvc;
1293
1294    template <typename V>
1295    void m_hv %= hvc;
1296
1297    template <typename V>
1298    void m_hv *= hvc;
1299
1300    template <typename V>
1301    void m_hv /= hvc;
1302
1303    template <typename V>
1304    void m_hv %= hvc;
1305
1306    template <typename V>
1307    void m_hv *= hvc;
1308
1309    template <typename V>
1310    void m_hv /= hvc;
1311
1312    template <typename V>
1313    void m_hv %= hvc;
1314
1315    template <typename V>
1316    void m_hv *= hvc;
1317
1318    template <typename V>
1319    void m_hv /= hvc;
1320
1321    template <typename V>
1322    void m_hv %= hvc;
1323
1324    template <typename V>
1325    void m_hv *= hvc;
1326
1327    template <typename V>
1328    void m_hv /= hvc;
1329
1330    template <typename V>
1331    void m_hv %= hvc;
1332
1333    template <typename V>
1334    void m_hv *= hvc;
1335
1336    template <typename V>

```

```

267     for (auto jt = std::next(it); jt != set.end(); ++jt) {
268         if (m_weakly_dominates(*it, aux)) {
269             set.erase(
270                 std::remove_if(jt, set.end(),
271                     [this, it](auto const& a) { return m_weakly_dominates(*it, a); }),
272                 set.end());
273             return;
274         } else {
275             std::swap(aux, *jt);
276         }
277     }
278     if (!m_weakly_dominates(*it, aux)) {
279         set.push_back(std::move(aux));
280     }
281 }
282 }
283
284 // limit_set (implementation)
285 template <typename S, typename V>
286 auto m_limit_set(S const& s, V const& v) const {
287     S res;
288     res.reserve(s.size());
289     for (auto const& p : s) {
290         auto aux = p;
291         for (size_t i = 0; i < aux.size(); ++i) {
292             aux[i] = std::min(aux[i], v[i]);
293         }
294         m_insert_non_dominated(std::move(aux), res);
295     }
296     return res;
297 }
298
299 template <typename V, typename R>
300 auto m_point_hv(V const& v, R const& r) const {
301     auto res = v[0] - r[0];
302     for (size_t i = 1; i < v.size(); ++i) {
303         res *= v[i] - r[i];
304     }
305     return res;
306 }
307
308 template <typename S, typename R>
309 auto m_set_hv3d(S const& s, R const& r) const {
310     using array2_t = std::array<hv_type, 2>;
311     auto aux = std::vector<array2_t>{{r[1], std::numeric_limits<hv_type>::max()},
312                                     {std::numeric_limits<hv_type>::max(), r[2]}};
313
314     hv_type v = 0;
315     hv_type a = 0;
316     hv_type z = 0;
317
318     for (auto const& p : s) {
319         v += a * (z - p[0]);
320         z = p[0];
321
322         auto tmp = array2_t{p[1], p[2]};
323         auto it = std::lower_bound(aux.begin(), aux.end(), tmp,
324             [](auto const& a, auto const& b) { return a[1] > b[1]; });
325         auto jt = it;
326
327         auto r0 = (*std::prev(it))[0];
328         auto r1 = tmp[1];
329         for (; (*it)[0] <= tmp[0]; ++it) {
330             a += (tmp[0] - r0) * (r1 - (*it)[1]);
331             r0 = (*it)[0];
332             r1 = (*it)[1];
333         }
334         a += (tmp[0] - r0) * (r1 - (*it)[1]);
335         if (jt != it) {
336             *jt = tmp;
337             aux.erase(++jt, it);
338         } else {
339             aux.insert(it, tmp);
340         }
341     }
342     v += a * (z - r[0]);
343     return v;
344 }
345
346 template <typename S, typename R>
347 auto m_set_hv(S const& s, R const& r, hv_type c = 1) const -> hv_type {
348     hv_type v = 0;
349
350     if (s.size() == 0) {
351         return 0;
352     }

```



```

357
358     if (s.begin()->size() == 2) {
359         hv_type r1 = r[1];
360         for (auto const& p : s) {
361             v += (p[1] - r1) * (p[0] - r[0]);
362             r1 = p[1];
363         }
364         v *= c;
365     } else if (s.begin()->size() == 3) {
366         v = c * m_set_hv3d(s, r);
367     } else {
368         auto newr = std::vector<hv_type>();
369         for (size_t i = 1; i < r.size(); ++i) {
370             newr.push_back(r[i]);
371         }
372         auto newl = std::vector<std::vector<hv_type>>();
373         newl.reserve(s.size());
374         for (auto const& p : s) {
375             auto newc = c * (p[0] - r[0]);
376             auto newp = std::vector<hv_type>();
377             for (size_t i = 1; i < p.size(); ++i) {
378                 newp.push_back(p[i]);
379             }
380
381             v += newc * m_point_hv(newp, newr) - m_set_hv(m_limit_set(newl, newp), newr, newc);
382
383             m_insert_non_dominated(std::move(newp), newl);
384         }
385     }
386     return v;
387 }
388
389 hv_type m_hv;
390 set_type m_set;
391 ovec_type m_ref;
392 };
393
394 } // namespace pmnk
395 #endif // WFG_H

```



# Index

/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/GSEMO/gsemo.hpp, [43](#), [44](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp, [45](#), [46](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ibea.hpp, [47](#), [48](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/PLS/pls.hpp, [60](#), [62](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/rMNKEval.hpp, [64](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/solution.hpp, [68](#), [69](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/Utils.hpp, [71](#), [72](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/wfg.hpp, [72](#), [77](#)  
/home/pedro/Documents/projects/anytime-pmnk-landscapes/src/main.cpp, [51](#)

add\_non\_dominated  
utils.hpp, [71](#)

CROSSOVER  
main.cpp, [53](#)  
Crossover  
main.cpp, [54](#)

decision\_vector  
pmnk::Solution, [34](#)  
dominance  
pmnk::Solution, [34](#)

eval  
pmnk::Solution, [35](#)

fitness  
pmnk::GASolution, [9](#)

GASolution  
pmnk::GASolution, [9](#)  
GSEMO  
pmnk::GSEMO, [11–14](#)  
gsemo  
main.cpp, [55](#)

IBEA  
pmnk::IBEA, [17–19](#)  
ibea  
main.cpp, [56](#)  
IHD  
pmnk::IHD< R >, [22](#)

INDICATOR  
main.cpp, [53](#)  
Indicator  
main.cpp, [55](#)  
insert\_non\_dominated  
wfg.hpp, [74](#)  
KWayTournamentSelection  
pmnk::KWayTournamentSelection< RNG >, [23](#)

limit\_set  
wfg.hpp, [74](#)

main.cpp  
CROSSOVER, [53](#)  
Crossover, [54](#)  
gsemo, [55](#)  
ibea, [56](#)  
INDICATOR, [53](#)  
Indicator, [55](#)  
MUTATION, [53](#)  
Mutation, [55](#)  
pls, [57](#)  
SELECTION, [54](#)  
Selection, [55](#)  
set\_general\_options, [57](#)  
set\_ibea\_crossover\_options, [58](#)  
set\_ibea\_mutation\_options, [58](#)  
set\_ibea\_options, [59](#)  
set\_ibea\_selection\_options, [59](#)  
set\_pls\_options, [59](#)  
set\_positional\_arguments, [60](#)

MUTATION  
main.cpp, [53](#)

Mutation  
main.cpp, [55](#)

neighborhood\_solutions  
pmnk::Solution, [35](#)  
non\_visited\_solutions  
pmnk::PLS, [30](#)  
NPointCrossover  
pmnk::NPointCrossover< RNG >, [25](#)

objective\_vector  
pmnk::Solution, [35](#)  
operator<<  
pmnk::Solution, [38](#)  
operator()  
pmnk::EPS, [7](#)  
pmnk::IHD< R >, [22](#)

- pmnk::KWayTournamentSelection< RNG >, 24
- pmnk::NPointCrossover< RNG >, 25
- pmnk::UniformCrossover< RNG >, 40
- pmnk::UniformMutation< RNG >, 41
- operator=
  - pmnk::Solution, 36
- operator[]
  - pmnk::Solution, 36
- PLS
  - pmnk::PLS, 27–29
- pls
  - main.cpp, 57
- pls.hpp
  - RUNLOOP, 61
- pmnk::EPS, 7
  - operator(), 7
- pmnk::GASolution, 8
  - fitness, 9
  - GASolution, 9
  - set\_fitness, 10
  - set\_objv, 10
- pmnk::GSEMO, 10
  - GSEMO, 11–14
  - run, 14
  - solutions, 14
- pmnk::hvobj< T >, 15
  - remove, 16
- pmnk::IBEA, 16
  - IBEA, 17–19
  - run, 20
  - solutions, 20
- pmnk::IHD< R >, 21
  - IHD, 22
  - operator(), 22
- pmnk::KWayTournamentSelection< RNG >, 23
  - KWayTournamentSelection, 23
  - operator(), 24
- pmnk::NPointCrossover< RNG >, 24
  - NPointCrossover, 25
  - operator(), 25
- pmnk::PLS, 26
  - non\_visited\_solutions, 30
  - PLS, 27–29
  - run, 30
  - solutions, 30
- pmnk::RMNKEval, 31
- pmnk::Solution, 32
  - decision\_vector, 34
  - dominance, 34
  - eval, 35
  - neighborhood\_solutions, 35
  - objective\_vector, 35
  - operator<<, 38
  - operator=, 36
  - operator[], 36
  - random\_solution, 37
  - size, 37
  - Solution, 33, 34
  - uniform\_bit\_flip\_solution, 37
- pmnk::UniformCrossover< RNG >, 39
  - operator(), 40
  - UniformCrossover, 39
- pmnk::UniformMutation< RNG >, 40
  - operator(), 41
  - UniformMutation, 41
- point\_hv
  - wfg.hpp, 75
- point\_hvc
  - wfg.hpp, 75
- random\_solution
  - pmnk::Solution, 37
- remove
  - pmnk::hvobj< T >, 16
- run
  - pmnk::GSEMO, 14
  - pmnk::IBEA, 20
  - pmnk::PLS, 30
- RUNLOOP
  - pls.hpp, 61
- SELECTION
  - main.cpp, 54
- Selection
  - main.cpp, 55
- set\_fitness
  - pmnk::GASolution, 10
- set\_general\_options
  - main.cpp, 57
- set\_hv
  - wfg.hpp, 76
- set\_hv\_wfg
  - wfg.hpp, 76
- set\_ibeas\_crossover\_options
  - main.cpp, 58
- set\_ibeas\_mutation\_options
  - main.cpp, 58
- set\_ibeas\_options
  - main.cpp, 59
- set\_ibeas\_selection\_options
  - main.cpp, 59
- set\_objv
  - pmnk::GASolution, 10
- set\_pls\_options
  - main.cpp, 59
- set\_positional\_arguments
  - main.cpp, 60
- size
  - pmnk::Solution, 37
- Solution
  - pmnk::Solution, 33, 34
- solutions
  - pmnk::GSEMO, 14
  - pmnk::IBEA, 20
  - pmnk::PLS, 30
- uniform\_bit\_flip\_solution

- pmnk::Solution, [37](#)
- UniformCrossover
  - pmnk::UniformCrossover< RNG >, [39](#)
- UniformMutation
  - pmnk::UniformMutation< RNG >, [41](#)
- utils.hpp
  - add\_non\_dominated, [71](#)
- weakly\_dominates
  - wfg.hpp, [77](#)
- wfg.hpp
  - insert\_non\_dominated, [74](#)
  - limit\_set, [74](#)
  - point\_hv, [75](#)
  - point\_hvc, [75](#)
  - set\_hv, [76](#)
  - set\_hv\_wfg, [76](#)
  - weakly\_dominates, [77](#)